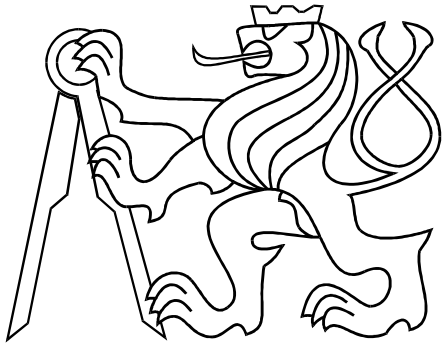
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ
KATEDRA KYBERNETIKY



DIPLOMOVÁ PRÁCE

Koordinace cíleného jednání skupiny mobotů

Pavel Novák
28. leden 2000

Prohlášení:

Prohlašuji, že jsem celou diplomovou práci včetně příloh vypracoval samostatně a uvedl jsem všechny použité podklady a literaturu.

Dále prohlašuji, že souhlasím s využitím této práce a jejích výsledků katedrou.

V Praze, dne 28. ledna 2000 ……..

Poděkování:

Na tomto místě bych chtěl poděkovat všem, kteří jakýmkoli způsobem přispěli k mému zdárnému dokončení této práce. Především bych chtěl poděkovat svým rodičům, za to že mi umožnili studovat na vysoké škole a přátelům, jmenovitě především Michaele Antlové, Dagmar Augustinové, Katce Eckmayerové, Jaroslavu Kočímu, Jiřímu Paloušovi a Jiřímu Pavlíčkovi, uvedeným v abecedním pořadí, kteří s ohleduplností snášeli především ve dnech posledních úprav této práce mé mnohdy zcela nesmyslné reakce, kterými jsem je v nejrůznějších situacích překvapoval. Specielně pak chci poděkovat svému spolubydlícímu Petru Svobodovi, který mi velmi pomohl v soustředění se na práci především tím, že tady většinu času nebyl. Také chci poděkovat neznámé dívce, jež mi ve stanici metra Národní třída věnovala nádherný usměv a dodala mi tak ve dnech mého váhání a přemýšlení nad smyslem této práce a života samotného novou energii, kterou jsem k dokončení této práce tolik potřeboval. Nemohu však také ještě nepoděkovat svým nervům za to, že v tomto těžkém období se mnou vydržely až dokonce, a že neselhaly ani v těch nejkritičtějších chvílích, v nichž si většina vegetativních funkcí mého organismu již počínala velmi nejistě. Děkuji …

Anotace:

Obsahem této práce je navržení řídícího systému mobilního robota (mobota), umožňujícího koordinaci jeho činnosti s činností jiného mobota, popř. celé skupiny. Jsou zde shrnuty základní přístupy distribuované umělé inteligence a pozornost je soustředěna především na reaktivní, chováním motivovaný, způsob řízení. Dále se práce věnuje možnostem neuronových sítí a především oblasti tzv. posilovaného učení (*Reinforcement learning*). Na tomto principu je též založena konečná implementace jedné z vrstev řídícího systému mobota, umožňující koordinaci na nejnižší úrovni. Tato vrstva je realizována pomocí asociativně vyhledávací neuronové sítě (Associative Search Network) a výsledky pomocí ní dosažené jsou demonstrovány prostřednictvím simulací uvedených v závěru této práce.

Contents of this work is the design of mobile robot control system enabling coordination of its action with the action of the other mobot, or the action of the whole group. There are summarized basic approaches of distributed artificial intelligence; it deals especially with behavioral control. The work also focuses on the possibilities of neural networks, primarily to the area of so called Reinforcement learning. Also the final implementation of one of the mobot control system layer, enabling coordination on the lowest level, is based on this principle. This layer is realized by Associative Search Network and the results achieved by it are demonstrated by the simulations stated in the end of this work.

Obsah

1 Úvod

Již od počátku věků postupuje člověk a lidská společnost ve svém vývoji rychlejšími kroky, než kterýkoli jiný živočišný druh na této planetě. Během několika málo tisíciletí své existence se lidský druh velmi výrazně odlišil od ostatních druhů, přítomných zde mnohonásobně déle, a s trochou předpojatosti si snad můžeme dovolit říci, že je ve svém vývoji předběhl. Lidský druh se nejen dokázal velmi rychle přizpůsobit životním podmínkám, které mu okolní prostředí poskytuje, ale velmi brzy si začal okolní prostředí upravovat tak, aby naopak právě životní podmínky přizpůsobil svým potřebám. I když je rozporuplné, zda je tato cesta opravdu správná, a zda právě tento způsob života, bráno z globálního pohledu, nepředurčuje lidstvo k záhubě, je více než jasné, že z pohledu časového úseku života jedince je tento směr vývoje výhodný. Ale mým záměrem není vyvolat diskusi nad tím, zda se lidstvo jako živočišný druh dalo na své cestě vývoje správným směrem, nýbrž dostat se k otázce z hlediska evoluce samotné mnohem podstatnější, tedy alespoň dle mého názoru, a tou je : „Co umožnilo lidstvu vydat se ve svém vývoji tak odlišným směrem ?“.

Co tedy je, nebo bylo, příčinou toho, že se lidstvo za tak krátkou dobu dostalo oproti ostatním živočišným druhům, na tak “vysokou“ úroveň ? Co mu umožnilo postupovat ve svém vývoji tak neuvěřitelně rychle ? A proč se v tak krátké době mohlo dostat do pozice “nadřazeného“ druhu, na jehož konání nyní již většina ostatních druhů obývajících tuto planetu přímo závisí, ať už to byl či nebyl osudný omyl přírody ?

Myslím, že stejně jako mně i Vás teď napadá jediná odpověď : „Inteligence !“. Právě svou inteligencí člověk ostatní formy života převyšuje, a právě díky ní dosáhl na této planetě tak významného postavení. Ale nespokojme se s tak jednoduchou a fádní odpovědí. Zamysleme se dál. Chceme snad tvrdit, že žádný jiný tvor této planety inteligentní není ? Samozřejmě že nechceme. Co tedy rozumíme pod tím tak známým a často omílaným pojmem jakým je „inteligence“ ?

Podle mého názoru, lze za inteligentního označit každého tvora, který je schopen přizpůsobovat se změnám prostředí ve kterém žije, a který je schopen cíleně ovlivňovat své chování za účelem uspokojení svých pudů, a to ať už těch nejprimitivnějších, jakými jsou např. hlad, nebo žízeň, a nebo těch nejabstraktnějších jakými jsou např. touha po vědění, nebo touha zkoumat a objevovat zákonitosti dění.

Možná by jste teď namítli, že to by pak byl „inteligentní“ téměř každý organismus, ale přesně tak to dle mého názoru taky je. A čím se tedy tak odlišujeme ? No právě že jen stupněm inteligence. Kromě uspokojování základních pudů nám totiž byl dán dar přemýšlet a co je mnohem podstatnější dar schopnosti abstrakce, fantazie a představivosti, které nám umožňují dar přemýšlet využívat smysluplným, efektivním a mnohdy až magickým způsobem. To je to co nás činí „inteligentnějšími“ a co nám umožnilo dát se směrem, kterým se žádný z ostatních druhů vydat nemohl, protože jejich fantazie a představivost nebyly rozvinuty do takové míry, aby jich mohly plně využít, aby mohly podobně jako my toužit po věcech, se kterými se nikdy přímo nesetkaly, a kterých my jsme dosáhli jen díky tomu, že se jakýmsi nepopsatelným způsobem zrodily ze střípků myšlenek v naší fantazii. To je to, co nám umožnilo rozšířit naše obzory za hranice momentální reality a co nám umožnilo nahlédnout, alespoň nepatrně, za okraj přítomnosti a naučit se tak čelit a především jít vstříc budoucnosti.

Dovolím si však tvrdit, že ani toto všechno by samo o sobě nestačilo v tom, udělat tak obrovský krok kupředu, nýbrž nám jen umožnilo se ujistit, že tento krok udělat lze. Samotné uvědomění si nějakého vyššího, a většinou o to vzdálenějšího, cíle, ve smyslu

ideálnějšího řešení daného problému, vize lepší budoucnosti apod., není totiž postačující. Naprosto nezbytný je totiž ještě způsob, kterým tento cíl lze dosáhnout. Je to sice "jen" prostředek, ale naprosto nezbytný, protože bez něj nikdy žádný pomyslný krok kupředu neuděláme, a náš cíl navždy zůstane jen snem, fantazií či myšlenkou, kterou nikdy nebudeme mít možnost změnit v realitu. A tímto "prostředkem" ve vývoji jakéhokoli druhu, tou zatím nepovšimnutou ale nejpodstatnější věcí, bylo objevení možností a síly spolupráce.

Při vytvoření skupiny spolupracujících jedinců totiž schopnosti této skupiny mnohonásobně převýší prostý součet všeho, čeho je schopen dosáhnout každý jedinec samostatně. Díky tomu i nepatrný rozdíl v organizovanosti určitých skupin, znamená obrovský rozdíl v jejich možnostech a schopnostech a tedy i v rychlosti jejich vývoje a vyspělosti. A to je ten pravý důvod, proč jsme se na rozdíl od ostatních živočišných druhů dostali na tak vysokou úroveň a co nám umožnilo ve vývoji postupovat tak rychle.

Stupeň naší inteligence nám totiž umožnil nejen mnohonásobně rozšířit naše obzory za hranice reality a objevit svět představ a fantazií, ale současně nám umožnil postupně vyvinout tak vysoce organizované společenství, které nám umožnilo začít tyto představy postupně naplňovat. A jen díky tomu jsme dosáhli svého postavení na této planetě, protože samotný jedinec, ať už s jakkoli dokonale vyvinutými představami, nikdy nemůže své okolí ve svém vývoji předstihnout nějak razantně, protože právě vyspělost jeho představ zamezuje tomu aby je mohl uskutečnit – sám.

Jak principy spolupráce, organizací, vzájemnými vazbami a vztahy mezi jednotlivci a hierarchií živočišných společenství, a to především lidských, tak čistě biologickými principy evoluce, adaptace a učení se nechává inspirovat *distribuovaná umělá inteligence*, které je věnována následující kapitola.

Pro větší přehlednost zde nyní ještě uvedu stručný nástin obsahu jednotlivých kapitol. Kapitola 2 - *Distribuovaná umělá inteligence* je tedy zaměřena na základní rozbor problematiky řešení úloh prostřednictvím součinnosti více systémů a především pak na shrnutí základních přístupů, prostředků a pojmů používaných v této oblasti. V kapitole 3 - *Výběr použitého přístupu* jsou uvedeny základní důvody, které mně vedly k orientaci na reaktivní, chováním motivovaný, způsob řízení a ke konečnému využití neuronové sítě jako prostředku pro řešení úlohy koordinace pohybu mobotů. Z tohoto důvodu jsou v kapitole 4 - *Neuronové sítě* rozebrány základní vlastnosti, možnosti a typy neuronových sítí, spolu s hlavními metodami používanými pro jejich učení (adaptaci). Kapitola 5 - *Existující implementace* je pak věnována několika existujícím projektům realizovaným ve světě, které mne určitým způsobem zaujaly, a které mne inspirovaly při návrhu řídícího systému mobota a konečné implementaci jedné z jeho vrstev, čímž se zabývá kapitola 6 - *Vlastní řešení*.

2 Distribuovaná umělá inteligence

Distribuovaná umělá inteligence (Distributed Artificial Intelligence – DAI) se zabývá součinností více systémů (agentů) při řešení společného problému. Snaží se tak využít nezměrných možností a výhod spolupráce, z nichž těmi nejpodstatnějšími jsou: zkrácení celkové doby řešení díky paralelnímu zpracování a zvýšení spolehlivosti a operativnosti z důvodu otevřenosti a granularity systému. Zabývá se především otázkami kompetencí jednotlivých agentů, způsobem a formalismem jejich vzájemné komunikace, metodami jejich koordinace a kooperace, tedy shrnuto, zabývá se organizací skupiny (společenství) spolupracujících agentů.

2.1 Přístupy DAI

Zmiňme se krátce a dvou hlavních přístupech DAI k řešení daného problému prostřednictvím více systémů, a to ať již systémů totožných a nebo různorodých, jakými jsou *distribuované řešení úloh* a *multiagentové systémy*.

2.1.1 Distribuované řešení úloh

Distribuované řešení úloh hledá pro pevně zvolenou úlohu způsob rozdělení práce a informací potřebných při jejím řešení mezi více systémů. Soustřeďuje se především na vytvoření efektivního plánu distribuovaného řešení dané úlohy, přičemž koordinace jednotlivých systémů často degraduje až na prostou pevně předdefinovanou synchronizaci. Takto vzniklá pevně provázaná skupina systémů je pak schopna řešit pouze přesně specifikovanou a velmi omezenou třídu úloh.

2.1.2 Multiagentové systémy

Multiagentové systémy kladou naopak důraz na autonomnost (samostatnost) jednotlivých systémů, spolupracujících v zájmu společného cíle. Tyto systémy jsou propojeny velmi volně a jsou právě z důvodu zdůraznění jejich samostatnosti označovány jako agenti. Multiagentové systémy se tedy nutně soustřeďují především na prostředky součinnosti jednotlivých agentů, neboť právě flexibilita jejich vzájemných vazeb pak umožňuje komunitě agentů řešit obecnější třídu úloh.

Problematika součinnosti agentů je zde dle předpokladů, na rozdíl od přístupu distribuovaného řešení úloh, mnohonásobně rozsáhlejší a složitější, neboť efektivita řešení dané úlohy, nebo dokonce její řešitelnost, závisí na úrovni metod dorozumívání mezi agenty, na metodách odhalování rozporuplných záměrů agentů, na možnostech agentů uvažovat o činnosti a plánech ostatních a na mnoha jiných faktorech.

„…různé procedury komunikace a interakce agentů přímo ovlivňují nutnou míru koordinace činnosti agentů, která zajišťuje soulad ve výsledném chování. Různé metody dekompozice úlohy a odtud odvozené vznikající podproblémy kladou rozdílné požadavky na druh interakce mezi agenty či na jejich schopnost uvažovat o chování ostatních agentů. Koordinace výsledného chování záleží na tom, jak se rozhoduje o neshodách v síti agentů, kteří agenti se na těchto rozhodnutích podílejí, …“, toliko z článku [Gasser, 1991].

2.2 Agent

Jako i v jiných relativně mladých oblastech vědních disciplín, ani zde přesná specifikace agenta jako takového neexistuje. Existuje sice mnoho definic a podrobných specifikací vlastností agenta, ale žádná z nich není a nemůže být zcela obecná jednoduše proto, že jsme oblast samotné organizace společenství jednak dostatečně neprozkoumali, ale především proto, že i to nejvyspělejší společenství bude vždy orientováno na řešení určitým způsobem zaměřené, třeba i velmi rozsáhlé, ale nikdy zcela obecné, třídy úloh.

Pánové Wooldridge a Jennings [Wooldridge, 1995] např. označují za agenta hardwarový, nebo častěji softwarový systém, který je *autonomní* (řídí své akce bez přímého zásahu člověka), *sociální* (interaguje s ostatními agenty), reaktivní (vnímá své okolí a reaguje na jeho změny) a *proaktivní* (reaguje nikoliv pouze v závislosti na svém momentálním stavu, ale cíleně, a to v tom smyslu, že může sám převzít iniciativu při řešení).

Místo hledání obecné definice agenta se nyní zaměříme na možné pohledy na jeho samotnou architekturu, abychom si tak lépe uvědomili různorodost možných přístupů a pochopili tak nemožnost jeho přesné a obecně platné specifikace.

Jednou z hlavních otázek, a to nejen z hlediska multiagentových systémů, je nutnost, nebo spíše výhodnost a přínos existence vnitřní reprezentace okolního prostředí. Dalším důležitým faktorem charakterizujícím agenta je pak úroveň mechanismů, na jejichž základě se rozhoduje mezi možnými variantami řešení svého cíle, přesněji do jaké míry využívá informací a znalostí o ostatních agentech. Než si tedy uvedeme rozdělení agentů z těchto dvou pohledů, uvědomme si, že nejsou zcela striktně oddělitelné a že se navzájem částečně prolínají.

2.2.1 Vnitřní reprezentace okolního prostředí

Na základě toho, zda agenti disponují vnitřní reprezentací okolního prostředí a využívají ji při své činnosti, si je můžeme, a vlastně i samotné multiagentové systémy, rozdělit na *kognitivní* a *reaktivní*. Je více než jasné, že každý z těchto způsobů realizace sám o sobě nemůže přinést zcela uspokojivé výsledky, a že ta správná cesta bude ležet někde uprostřed, v jejich vzájemné kombinaci. Žádný z kognitivních systémů se totiž neobejde bez striktně reaktivních vazeb, jakými jsou např. reakční vazby řešící konfliktní situace na nejnižší úrovni (při nárazu na překážku zastav apod.) a na druhou stranu čistě reaktivním systémem nikdy nebudeme moci dosáhnout koordinace či kooperace agentů na vyšší úrovni, což je, jak již jsme si snad dostatečně objasnili, v současném vývoji hlavním cílem.

2.2.1.1 Kognitivní multiagentní systémy

Kognitivní agent využívá při řešení daného cíle vnitřní symbolickou explicitní reprezentaci okolního prostředí, na jejímž základě je schopen predikovat budoucí události a přizpůsobovat jim své chování. Je řízen cíleně, tj. explicitními cíli, které řídí jeho chování a dávají mu možnost volit mezi možnými akcemi [Ferber, 1996]. To do značné míry rozšiřuje jeho schopnosti, ale zároveň klade vyšší nároky na implementaci jeho metod řízení. Navíc musí být agent schopen svoji vnitřní reprezentaci prostředí dostatečně dynamicky aktualizovat.

2.2.1.2 Reaktivní multiagentní systémy

Reaktivní agent užívá chování typu podnět-odezva, tj. reaguje na současný stav prostředí ve kterém se nachází [Ferber, 1996]. Implementace takovýchto agentů je mnohem snazší než u agentů kognitivních, ale nepřináší možnosti interakce agentů na vyšší úrovni. Nicméně právě jednoduchost a tedy snadnost jejich vzájemné interakce je tou vlastností, díky níž je reaktivním agentům věnována taková pozornost. Možnosti jejich interakce totiž mohou vést k vytvoření, z globálního pohledu multiagentního systému, velmi složitých modelů chování, značně převyšujících možnosti jednoduché implementace vlastního agenta (takzvané emergentní chování).

Hojnost vzájemných interakcí je tímto způsobem schopna do jisté míry "konkurovat" možnostem složitějších metod užívaných při koordinaci či kooperaci kognitivními systémy, nicméně jak jsem již uvedl, tou správnou cestou bude, dle mého názoru, teprve kombinace reaktivního a kognitivního přístupu.

2.2.2 Volba varianty dosažení cíle

Podle úrovně schopností rozhodovat se mezi různými variantami řešení svého cíle můžeme agenty rozdělit do tří kategorií, a to na *reaktivní*, *intencionální* a *sociální* [Mařík, 1997].

2.2.2.1 Reaktivní agent

Reaktivní agent reaguje výhradně na podněty z okolního prostředí, a to jednou z množiny přesně definovaných akcí. To zda určitá akce ze známé množiny bude provedena, závisí na tom, zda stav prostředí, tak jak je toto prostředí agentem vnímáno, odpovídá podmínkám pro její zahájení. Reaktivní agent také může na základě podnětů z prostředí upřesňovat svůj obraz světa. Jelikož podněty z okolního prostředí jsou jedinou informací, kterou agent při svém jednání využívá, jsou jeho schopnosti výrazně závislé na volbě souboru jeho čidel (množiny jeho vjemů).

2.2.2.2 Intencionální (deliberativní) agent

Intencionální agent při volbě varianty řešení již zvažuje své možnosti dosažení daného cíle. Vytváří si plán akcí, ve kterém se silně odráží jeho motivace, to čemu věří a to čemu momentálně věří a plánují ostatní agenti. Agenti se o svých plánech samozřejmě vzájemně informují.

2.2.2.3 Sociální agent

Sociální agent již dokonce pracuje s modely chování ostatních agentů. To však sebou přináší nutnost schopnosti tyto modely aktualizovat a tedy další, nezanedbatelné, nároky na získávání informací jak z prostředí, tak od ostatních agentů.

2.3 Komunikace

Hlavním přínosem vytváření živočišných společenství, a jimi inspirovaných distribuovaných systémů, jsou bezbřehé výhody spolupráce, tedy vzájemné koordinace a kooperace jejich členů. K tomu, aby vůbec byla koordinace či kooperace možná, musí mít jednotliví agenti ve větší či menší míře k dispozici informace o ostatních členech společenství (multiagentového systému). A právě zprostředkování těchto informací zajišťuje komunikace. Vlastnosti distribuovaného systému jsou na způsobu komunikace zcela závislé.

Způsoby komunikace můžeme rozdělit především podle cíle zpráv zasílaných ve společenství agentů, a to na komunikaci *přímou* a *nepřímou* [Mařík, 1997].

2.3.1 Přímá komunikace

Při tomto typu komunikace mohou jednotliví agenti zasílat zprávy pouze přímo dalším agentům. Tento způsob komunikace lze dále rozdělit podle počtu adresátů zasílané zprávy na *adresné*, *všesměrové* a *selektivní* zasílání zpráv.

2.3.1.1 Adresné posílání zpráv

Při adresném posílání zpráv má každá zpráva jednoho, *adresou* přesně specifikovaného, příjemce. Tento způsob výrazně snižuje počet vyměňovaných zpráv a umožňuje též efektivní ošetření z hlediska bezpečnosti, ale na druhou stranu musí každý agent znát adresy všech agentů se kterými si potřebuje vyměňovat informace, a to zejména při budování otevřených systémů přináší nemalé problémy.

2.3.1.2 Všesměrové vysílání zpráv

Všesměrové vysílání zpráv je způsob komunikace, při kterém je každá jednotlivá zpráva zaslána najednou všem agentům. To přináší nesporné výhody z hlediska možnosti naprosté otevřenosti systému, z hlediska odolnosti systému vůči poruchám (poškození agenti mohou být nahrazeni jinými, bez nutnosti dalších zásahů do systému), ale oproti tomu znemožňuje bezpečnostní ochranu, jelikož každý agent má přístup k obsahu všech zpráv, a také může množstvím zasílaných zpráv nadměrně zatěžovat jak komunikační infrastrukturu tak samotné agenty.

2.3.1.3 Selektivní vysílání zpráv

Při selektivním vysílání zpráv jsou agenti rozděleni do skupin a jednotlivé zprávy jsou zasílány současně vždy všem členům jedné nebo více skupin. Tento způsob komunikace tak představuje kombinaci komunikace adresné a všesměrové. Každý agent přitom může být členem několika skupin současně. Tímto způsobem lze využívat výhod všesměrového vysílání při potlačení jeho hlavních nevýhod, tak jak byly uvedeny v předcházejícím odstavci.

2.3.2 Nepřímá komunikace

Při tomto typu komunikace je předávání zpráv realizováno prostřednictvím předem zvolené struktury (obvykle nazývané *tabule*), která svým charakterem odpovídá sdílené paměti.

2.3.2.1 Tabule

Tabulí je nazývána sdílená datová struktura, do níž jednotliví agenti zasílají údaje o průběhu své činnosti ve formě mezivýsledků a výsledků, kterých dosáhli. Naopak z této tabule získávají informace, o činnosti ostatních agentů a o momentálním stavu řešení dané úlohy, které je zajímají. Komunikace pomocí tabule je tedy realizována zasíláním zpráv na jedinou adresu (adresu tabule) a současně obracením se na tuto adresu jako na jediný zdroj informací.

Tento způsob komunikace umožňuje efektivní výměnu množství informací současně se snížením zatížení komunikační infrastruktury. Navíc umožňuje vytvářet a používat složité struktury organizace agentů.

Z pohledu distribuovaného zpracování znalostí (a ne jen prosté komunikace) můžeme agenty, kteří modifikují obsah tabule, pokud jsou na základě určitého

předchozího stavu tabule aktivováni, považovat za *zdroje znalostí*. Tabule pak obsahuje ještě tzv. řídící část, která určuje způsob dalšího postupu řešení, pokud na momentální stav datové struktury tabule zareaguje více zdrojů znalostí (agentů) najednou. Podrobněji viz [Mařík, 1997].

2.4 Koordinace a kooperace

Základní prostředky koordinace, jakými jsou závazky a smlouvy, a kooperace, jakými jsou vyjednávání a vícestupňové vyjednávání, jsou velmi srozumitelně rozebrány v knize [Mařík, 1997] a zde se jimi podrobněji zabývat nebudeme, protože řešení úlohy této práce metody koordinace a kooperace na tak vysoké úrovni využívat nebude.

Zamysleme se tedy alespoň nad významem obou pojmů a ujasněme si jejich základní principy.

2.4.1 Koordinace

Koordinace vytváří mezi agenty takové vztahy a vazby, které jim umožňují, aby si při plnění svých vlastních cílů nepřekáželi a navzájem se neomezovali, a nebo tak činili alespoň co nejméně. Typickým příkladem může být sdílení společných zdrojů více agenty, nebo situace, kdy se dva moboti budou najednou snažit projet úzkými dveřmi. Nejjednodušším způsobem koordinace by tedy mohlo být sekvenční seřazení akcí všech agentů, což by ale samotné společenství zcela degradovalo, a postavilo ho tak na úroveň řešení jediným agentem. Metody koordinace tedy musí zajišťovat bezkonfliktní fungování agentů celého multiagentového systému a přitom zachovat jeho paralelní charakter zpracování řešení.

2.4.2 Kooperace

Kooperace se oproti koordinaci zabývá mnohem složitějším problémem, a tím je přímá spolupráce skupiny agentů na řešení určité úlohy. Jejím úkolem je vytvořit mezi agenty spolupracující skupiny vztahy a vazby nikoli zajišťující pouze jejich bezkonfliktní chování, ale zajišťující jejich přímou součinnost. Příkladem může být manipulace břemene za pomocí více agentů apod. Metody kooperace jsou tedy podstatně náročnější, ale teprve ony mohou přinést ty nejhodnotnější výhody multiagentního zpracování, výhody spolupráce, tak jak jsme se o nich zmiňovali v úvodu.

3 Výběr použitého přístupu

Jak jsem již uvedl v předchozí kapitole, je jednou z hlavních otázek, a to nejen z hlediska multiagentových systémů, výhodnost a přínos existence vnitřní reprezentace okolního prostředí. Multiagentové systémy jsme si dle toho, zda vnitřní reprezentaci okolního prostředí využívají, rozdělili na *kognitivní* a *reaktivní* a současně jsme se snad shodli i na tom, že každý z těchto způsobů realizace sám o sobě nemůže přinést zcela uspokojivé výsledky, a že ta správná cesta bude ležet někde uprostřed, v jejich vzájemné kombinaci.

Ano, vnitřní reprezentace okolního prostředí je opodstatněná a přináší nesporné výhody, z nichž hlavními jsou možnosti realizace prostředků součinnosti agentů na vyšší úrovni, nicméně postupovat se má od jednoduššího ke složitějšímu, není-liž pravda ? Pokusme se tedy zaměřit raději na mnohem snadnější implementaci prostředků součinnosti, jaké nám nabízí reaktivní přístup.

Dle mého názoru prohloubením znalostí v oblasti reaktivního multiagentního systému, velmi výrazně rozšíříme své schopnosti následně smysluplně a efektivně využít prostředků a možností přístupu kognitivního. Ideálním případem by pak bylo, dospět ve vzájemné součinnosti agentů reaktivního multiagentního systému na takovou úroveň, kdy by se zcela zřejmým krokem kupředu stalo právě rozšíření agentů o znalost určitých zákonitostí prostředí ve kterém se pohybují, způsoby získávání těchto znalostí a jejich zpracování, a tedy plynulý přechod na určitý hybrid reaktivního a kognitivního modelu. V žádném případě však nepopírám, že stejného a možná i lepšího výsledku lze dosáhnout opačným způsobem, tedy směrem shora od systémů kognitivních.

Z tohoto důvodu jsem tedy řešení úlohy této práce orientoval směrem k reaktivním přístupem inspirované implementaci. Nicméně, jak později uvidíme, vlastní řešení, využívající možností neuronové sítě, tak zcela čistě reaktivní již není a snad mohu s trochou euforie říci, že je právě tím zmíněným, třebaže malinkým, krůčkem, směrem k zmíněnému "reaktivně-kognitivnímu" modelu. Ale nepředbíhejme a vraťme se ještě o něco podrobněji k reaktivním multiagentovým systémům.

3.1 Architektura reaktivních agentů

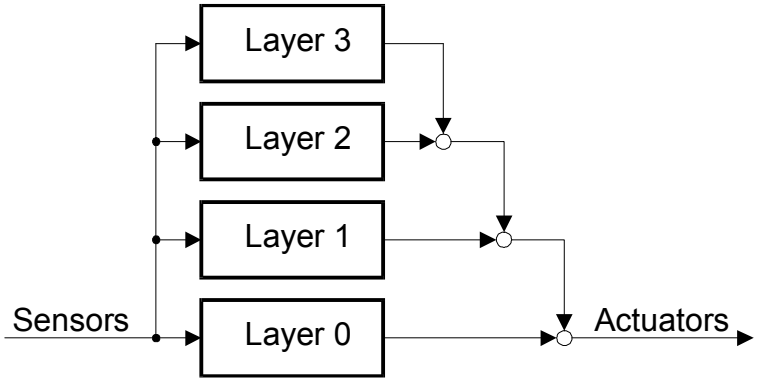
Než se zaměříme na bližší popis základních principů použité realizace řešení, tedy na stručný úvod do neuronových sítí obecně, uvedeme si zde neuronové sítě, a to zcela záměrně, nejprve pouze jako jednu z možných implementací reaktivních agentů, a to proto, abychom si tak lépe uvědomili jejich výhody a zařazení.

3.1.1 Situační pravidla

Situační pravidla (Situated rules), jak bychom mohli volně přeložit, jsou nejjednodušší architekturou modelu chování reaktivního agenta. Agent ke své činnosti využívá konečný počet pravidel, které při daném stavu okolního prostředí, tak jak ho momentálně agent vnímá svými senzory, přesně určují akci z množiny definovaných akcí, kterou je třeba provést. Agent nemá žádnou informaci o minulých stavech a jeho reakce je závislá pouze na současném vjemu prostředí. Chování agenta je tedy realizováno konečným počtem přesně definovaných odezev na přesně definované vstupy. Implementace této architektury je velmi snadná, ale velkou nevýhodou je to, že se všechna pravidla musí navzájem vylučovat a nesmí obsahovat žádný konflikt, což jejich formulaci výrazně ztěžuje a mnohdy i zcela znemožňuje.

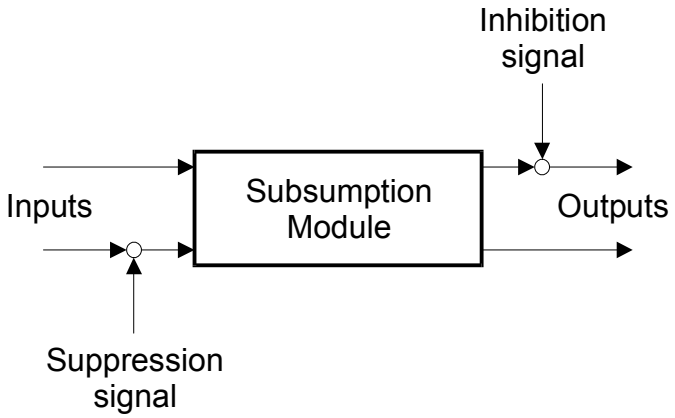
3.1.2 Vrstvená architektura

Vrstvená architektura (Subsumption Architecture), navržená a prezentovaná v roce 1986 R.A. Brooksem, je založena na rozdělení řídícího systému agenta do vrstev, viz obr. 1, jež odpovídají různým úrovním modelu chování agenta. Vyšší vrstvy staví na vrstvách nižších a vytvářejí tak mnohem komplexnější reakce a způsoby chování [HTTP, University of Michigan].



obr. 1 - Vrstvená architektura - blokové schéma

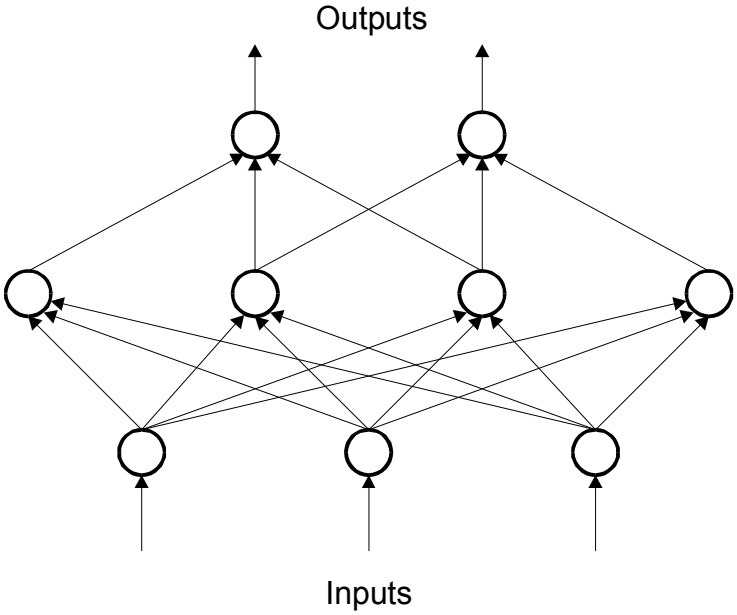
Jednotlivé vrstvy jsou tvořeny sítí rozšířených konečných stavových automatů (dále již jen automatů). Každý automat má speciální vstupní a výstupní signál. Pokud hodnota vstupního signálu překročí stanovenou mez, je chování implementované tímto automatem aktivováno. Automat, viz obr. 2, akceptuje navíc tzv. *suppression* signál, který přepisuje hodnotu jeho vstupního signálu, a *inhibition* signál, který naopak zcela potlačuje signál výstupní. Tím je umožněno vzájemné převažování jednotlivých druhů chování. Vstupy jednotlivých automatů tedy pochází ze senzorů agenta (vjemy), nebo z jiných automatů a jejich výstupy ovlivňují přímo akční členy, nebo jiné automaty. Velmi podrobné informace lze nalézt na [URL, Garforth].



obr. 2 - Zahrnutí "s*uppression*" a "*inhibition*" signálů

3.1.3 Neuronové sítě

Architektura neuronové sítě (NS), je sama o sobě také strukturou distribuovaného paralelního zpracování dat. Je tvořena množstvím elementárních jednotek, tzv. neuronů, které jsou navzájem různě pospojovány, viz obr. 3. Jednotlivé neurony se nevyznačují nijak vysokou složitostí, zjednodušeně řečeno pouze aktivují svůj výstup na základě toho, zda součet hodnot jejich vstupů překročí stanovenou mez.



obr. 3 - Příklad architektury NS

To co neuronovým sítím dává tak obrovské možnosti a přednosti, jakými jsou schopnosti učení, adaptace a především zobecňování či generalizace, je právě struktura a složitost vysoké vzájemné propojenosti jednotlivých neuronů, ale o tom až v kapitole 4 - *Neuronové sítě*. Výhodou této architektury jsou mnohem komplexnější reakce agenta a především výrazně plynulejší přechody mezi jednotlivými způsoby jeho chování, nevýhodou je určitá neprůhlednost konkrétní specifikace pravidel, na jejichž základě vlastní rozhodování probíhá. Agent totiž předem nemá přesně definovány jednotlivé vazby mezi vstupy čidel (vjemy) a příslušnými odezvami (akcemi), ale tyto vztahy a vazby si během fáze učení vytváří sám (blíže viz kapitola 4).

3.2 Souvislost s etologií

Jak jsme se dočetli v zadání této práce, měl bych se při jejím řešení v maximální míře opírat o poznatky etologie, a jak lze jistě podotknout, ještě zde o ní nepadlo ani slovo. Ano o samotné etologii jako vědě, jsem se zde zatím nezmínil, ale je zcela zřejmé, že vše co jsme si zatím uvedli s etologií přímo souvisí. Jak již víme, distribuovaná umělá inteligence, která je na pozadí všech přístupů zde uváděných, se nechává inspirovat principy spolupráce, organizací, vzájemnými vazbami a vztahy mezi jednotlivci, hierarchií živočišných společenství a stejně tak i čistě biologickými principy evoluce, adaptace a učení. DAI však tyto oblasti sama nezkoumá, nýbrž využívá poznatků vědy, která se chováním živočichů zabývá cíleně, a touto vědou je právě etologie.

Uveďme si nejprve definici etologie, tak jak ji uvádí *Akademický slovník cizích slov* [Buchtelová, 1998]:

„etologie, -e ž<ř> nauka zabývající se životními projevy a chováním živočichů“.

Zakladatel etologie, přírodovědec Konrad Lorenz, v historickém úvodu (str. 13) své knihy *Základy etologie* [Lorenz, 1993] píše: „*Etologie* neboli *srovnávací výzkum chování* je snadno definovatelná. Spočívá v tom, že se na chování zvířat a člověka aplikují všechna ta pojetí otázek a všechny ty metody, které se od Darwinovy doby staly ve všech jiných odvětví biologie samozřejmými.“.

Dierk Franck v knize *Etologie* [Franck, 1996] na str. 9 uvádí: „Etologie je věda zabývající se srovnáváním chování živočichů a dále i člověka z pozic biologie a biologickými metodami. Chování spočívá na organizačních schopnostech zvířat, které se uplatňují na různých integračních rovinách, např. na rovině molekulárních nebo biochemických procesů nebo na úrovni výzkumu smyslových orgánů, nervové soustavy a systému různých hormonů (humorální systém). Etolog pracuje z nadhledu, na rovině, ležící nad rovinami právě uvedených systémů.“

Jsem si vědom, že poslední dvě uvedené citace, nemusí být, z důvodu vytržení z kontextu a neobjasnění tak pojmů a myšlenek v nich uvedených, zcela srozumitelné, ale uvedl jsem je jen z důvodu nastínění složitosti a komplexnosti problémů, jimiž se etologie zabývá. Jinak nám pro rozsah této práce bude stačit definice prvá, která nám otázku tohoto odstavce, tedy souvislosti etologie a principů použitých v této práci, dostatečným způsobem zodpovídá.

Konkrétními mechanismy chování z pohledu etologie, jakými jsou např. dědičně koordinované neboli instinktivní pohyby a pojmy jako např. vrozený spouštěcí mechanismus, apetenční chování apod., si zde popisovat nebudeme, jelikož se jedná o rozsáhlou oblast, jejíž prostudování sice bylo základem zformování mého pohledu na oblast DAI a mobilní robotiky jako takové, ale která svým rozsahem zdaleka překračuje možnosti této práce. Z tohoto důvodu tedy “pouze” odkazuji především na knihy [Lorenz, 1993] a [Franck, 1996], které se danou problematikou zabývají velmi podrobně.

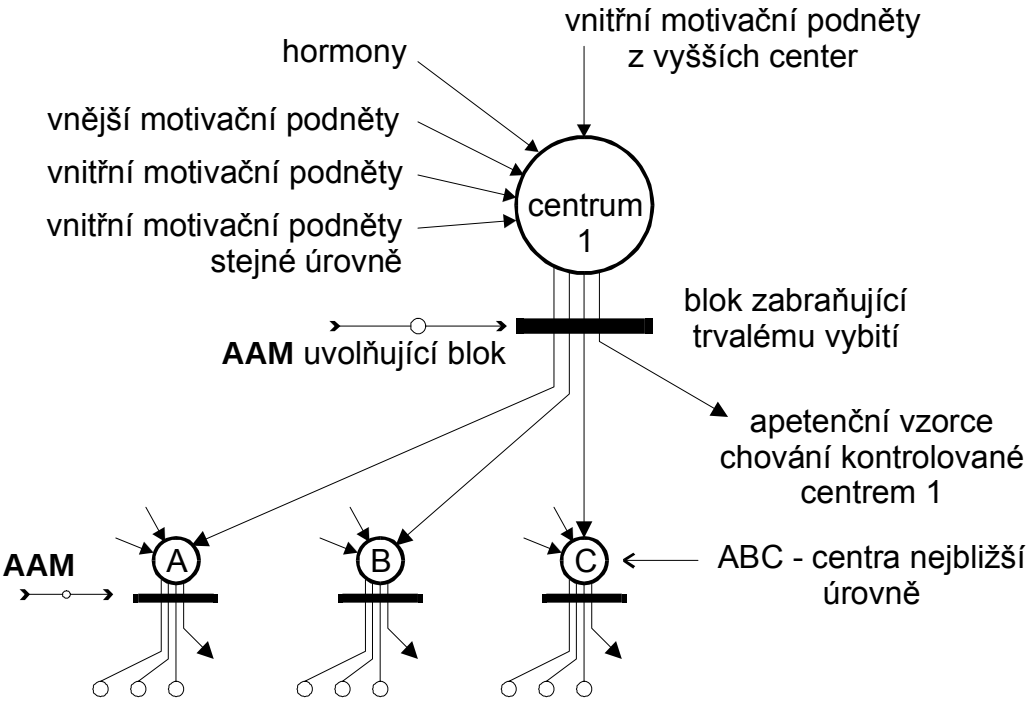
3.3 Důvody výběru zvoleného přístupu

Na začátku této kapitoly jsem se pokusil objasnit, proč jsem vlastní řešení zadané úlohy orientoval směrem k reaktivním způsobem řízení inspirované implementaci. Také jsem se již zmínil, že tím pravým a smysluplným řešením, které může ve vývoji distribuované umělé inteligence, a to jak z pohledu společenství spolupracujících agentů, tak z pohledu samostatného a zcela individuálního jedince, být výrazným krokem kupředu, je podle mého názoru určitá kombinace reaktivního a kognitivního (současnými metodami implementovaného) přístupu. A vydat se touto “zlatou střední cestou“ by nám mohly umožnit právě zmíněné neuronové sítě, jejichž možnosti realizovat komplexní reaktivní vazby a současně uchovávat informaci o prostředí (tj. vytvářet jeho reprezentaci) jsou zatím, a to pouze z důvodu nedostatečných výpočetních prostředků, nedoceněny.

Tím hlavním důvodem, tím co výrazně ovlivnilo způsob mého řešení tématu této práce, a co neustále ovlivňuje a nejspíše i nadále bude ovlivňovat mé konání, a to v jakékoli oblasti, je můj respekt, obdiv a ocenění smysluplnosti, propracovanosti, promyšlenosti, obsáhlosti a rozmanitosti, složitosti, robustnosti a efektivity systémů, kterým dala vzniknout evoluce během miliard let existence života na této planetě.

Je zcela zřejmé, že hledat inspiraci u živých organismů je i při současném stupni vývoje stále tou nejlepší zvolenou cestou.

Bez potřebných detailů a bližších podrobností, zde nyní uvedu obrázek hierarchického modelu instinktivního chování živočichů podle N. Tinbergena [Tinbergen, 1951], převzatý z [Lorenz, 1993] str. 136, který přibližuje tzv. CEM-AAM systém, tedy centrálně podněcující (CEM) a vrozený spouštěcí mechanismus (AAM), popisující procesy, k nimž dochází v nervové soustavě organismů v souvislosti s instinktivním a apetenčním chováním. Uveďme jen, že tento obrázek (obr. 4), velmi zjednodušeně řečeno, znázorňuje mechanismus aktivace určitého způsobu chování živočichů, a že tzv. vrozený spouštěcí mechanismus (AAM), zprostředkovávající zvířeti poznání biologicky relevantní situace v prostředí, můžeme z našeho pohledu brát jako pouhý vstup informace z prostředí (vjem). Bližší informace lze nalézt ve zmíněných zdrojích.



obr. 4 - Hierarchický model instinktivního chování

Podobnost architektury tohoto modelu chování živočichů s příkladem architektury neuronové sítě uvedené v odstavci 3.1.3, a především pak výrazná podobnost struktury jeho elementárních prvků se strukturou neuronů, která bude podrobněji rozebrána v kapitole 4 - *Neuronové sítě*, byla dalším podstatným důvodem, který mně k využití reaktivního, chováním motivovaného řídícího systému a k využití neuronové sítě jako prostředku implementace jeho vrstvy koordinace, vedl. Tolik tedy k vlastním důvodům výběru mnou zvoleného přístupu.

4 Neuronové sítě

Neuronové sítě (Neural Networks) se snaží matematickými metodami simulovat procesy probíhající v lidském mozku, a to nejen samotný způsob zpracovávání a uchovávání informací, ale především schopnost v těchto informacích rozpoznávat podobnosti. Samotná struktura mozku odpovídá značně složitě pospojované multiprocesorové architektuře.

Za umělou neuronovou síť je tedy obecně považována struktura určitým způsobem vzájemně pospojovaných výkonových prvků (neuronů), umožňující distribuované paralelní zpracování dat. Takováto síť pak zpracovává předložené vstupy a výsledek zpracování předkládá prostřednictvím jednoho nebo více výstupů. Neuronová síť tedy z tohoto pohledu transformuje vstupní signál $\vec{x}$ na výstupní signál $\vec{y}$ dle vztahu:

$$\vec{y} = T(\vec{x}),$$

kde

$\vec{x}$ … je vektor vstupů,

$\vec{y}$ … je vektor výstupů,

T … je transformační funkce NS.

Podle Kolmogorova teorému je neuronová síť (NS), která má alespoň tři vrstvy o odpovídajících počtech neuronů, schopna aproximovat zcela libovolnou přenosovou funkci. Bohužel důkaz tohoto teorému má pouze existenční charakter, tedy je pouze známo, že daná neuronová síť existuje, ale neví se jak ji obecně vytvořit.

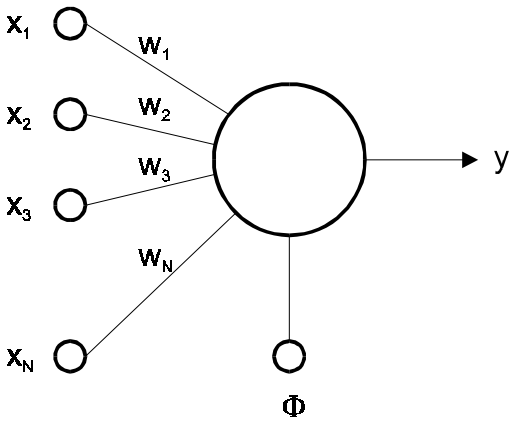
V současné době existuje mnoho druhů neuronových sítí nejrůznějších struktur (jednovrstvé, vícevrstvé, plošné, prostorové, mnohorozměrné, apod.), které jsou však většinou zaměřeny na řešení specifických úloh, a to především v oblastech, jakými jsou klasifikace, predikce, filtrace, aproximace a mnohé další.

4.1 Historický vývoj

Prvopočátkem umělých neuronových sítí bylo navržení prvního matematického modelu neuronu v roce 1943. Učinili tak pánové Warren S. McCulloch a W. Pitts [McCulloch, 1943]. Formulace metody Hebbova učení [Hebb, 1949] a Rosenblattův perceptron (rozšíření neuronu) [Rosenblatt, 1958] a [Rosenblatt, 1962] byly dalšími podstatnými kroky kupředu v této oblasti. V práci [Minsky, 1969] se pánové M. Minsky a S. Papert velmi podrobně zabývali problematikou perceptronu, poukázali na jeho nedostatky a podložili svá tvrzení matematickými důkazy. Vlivem této práce na určitou dobu zájem o neuronové sítě utichl. Důvodem návratu k neuronovým sítím a počátkem jejich dalšího rapidního rozvoje byly pak práce [Hopfield, 1982] a především pak článek o učení pomocí zpětné propagace chyb [Rumelhart, 1986], který pánové D.E. Rumelhart, G.E. Hinton a R.J. Williams uveřejnili roku 1986. V devadesátých letech tak započala nová epocha bádání v oblasti neuronových sítí a vznikla obrovská spousta upravených, ale i zcela nových struktur a modelů NS.

4.2 Neuron

Neuron je základním stavebním prvkem neuronových sítí a jeho základní struktura je uvedena na obr. 5. Každý neuron má jeden výstup, a volitelný počet vstupů. Hodnotami vstupů neuronů jsou vjemy z prostředí (zprostředkované senzory), nebo aktivace (tj. hodnoty výstupů) jiných neuronů. Ke každému vstupu je přiřazena tzv. váha, která určuje vliv toho vstupu na výstup neuronu. Výstup neuronu je aktivován, pokud vážený součet jeho vstupů překročí stanovenou prahovou hodnotu.



obr. 5 - Struktura neuronu

Výstup neuronu y tedy odpovídá vztahu:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + \Phi\right),$$

kde

x_i … je i-tý vstup neuronu,

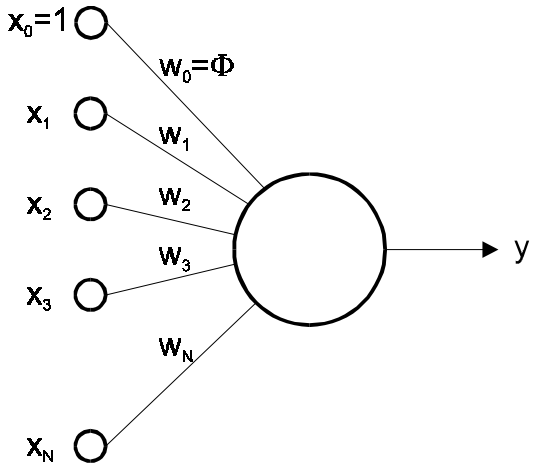
w_i … je váha i-tého vstupu neuronu,

N … je počet vstupů neuronu,

Φ … je práh neuronu,

f … je aktivační (přenosová) funkce neuronu.

Velmi často se práh neuronu realizuje jako další (nultý) vstup, jehož hodnota je trvale nastavena na hodnotu jedna a jehož váha je rovna právě hodnotě požadovaného prahu neuronu. Struktura neuronu pak odpovídá obr. 6 a samotný výpočet výstupu neuronu se tak zjednoduší. Tato zdánlivě nepodstatná úprava je velkým přínosem při implementaci neuronových sítí výpočetními prostředky, jelikož se tak při obvykle značně velkém počtu neuronů sítě, výrazně sníží výpočetní a časová náročnost algoritmu vybavování.



obr. 6 - Struktura neuronu (práh jako jeden ze vstupů)

Výstup neuronu y pak tedy odpovídá vztahu:

$$y = f\left(\sum_{i=0}^{N} w_i x_i\right),$$

kde

x_i … je i-tý vstup neuronu,

w_i … je váha i-tého vstupu neuronu,

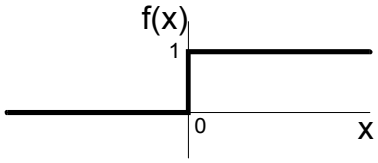
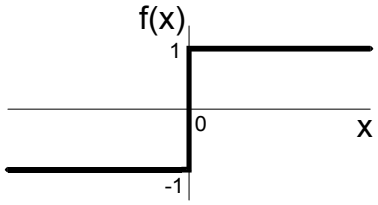
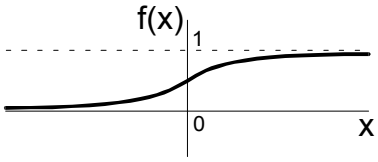
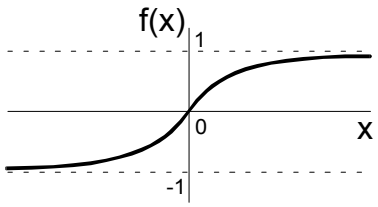
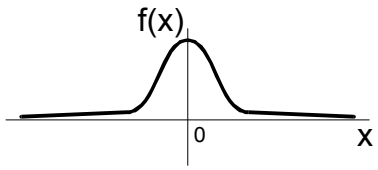
N … je počet vstupů neuronu,

f … je aktivační (přenosová) funkce neuronu.

4.2.1 Aktivační funkce neuronu

Aktivační (přenosovou) funkcí neuronu je zpravidla nějaká nelineární funkce. U prvního modelu neuronu [McCulloch, 1943] to byla funkce jednotkového skoku (unit step function), pro diskrétní neuronové sítě je vhodná zejména skoková funkce (step function) a pro spojité neuronové sítě jsou nejpoužívanějšími funkcemi logistická funkce (logistic function), tanh a Gausián (Gaussian). Příklady aktivačních funkcí neuronů jsou uvedeny v tab. 1.

Nelinearita aktivační funkce neuronů skrytých vrstev vnáší velmi důležitou a potřebnou nelinearitu do celé neuronové sítě. Bez nelinearity by totiž skryté vrstvy NS nebyly zdaleka tak obrovským přínosem, jelikož právě nelineární charakter, tj. schopnost aproximovat nelineární funkce, činí vícevrstvé NS tak výkonnými.

Unit Step		$f(x) = \begin{cases} 0 & if \quad x < 0 \\ 1 & if \quad x \geq 0 \end{cases}$
Step		$f(x) = \begin{cases} 1 & if \quad x > 0 \\ f_{t-1}(x) & if \quad x = 0 \\ -1 & if \quad x < 0 \end{cases}$
Logistic		$f(x) = \frac{1}{1+e^{-x}}$
tanh		$f(x) = \tanh(x)$
Gaussian		$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

tab. 1 - Aktivační funkce neuronu (příklady)

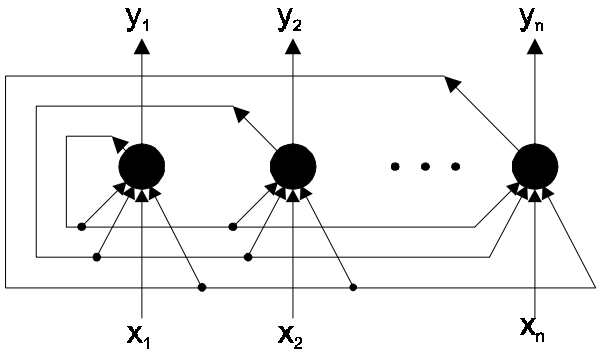
4.3 Topologie neuronové sítě

Jak jsme si již řekli v úvodu této kapitoly, neuronová síť je tvořená množstvím elementárních prvků (neuronů), které jsou navzájem určitým způsobem pospojovány. Topologie neuronové sítě určuje celkovou strukturu vzájemného propojení jednotlivých neuronů a bývá popsána grafem, jehož uzly odpovídají neuronům a hrany spojům mezi nimi. Vstupem neuronu může být výstup senzoru (vjem z prostředí), výstup jiného

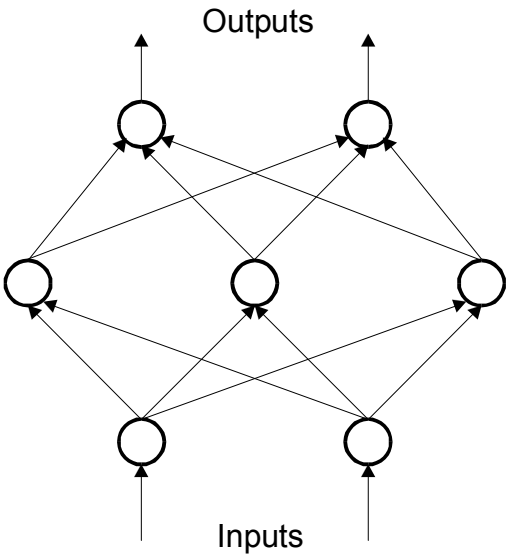
neuronu, a nebo i vlastní výstup tohoto neuronu. Výstup neuronu pak může vést jako vstup do jiného neuronu (včetně sebe sama), a nebo být přímo výstupem neuronové sítě. NS tak mohou vytvářet nejrůznější struktury, z nichž nejpoužívanější jsou jednovrstvé, vícevrstvé a plošné.

Kromě pohledu prostorového uspořádání se NS mohou lišit ještě způsobem šíření signálu. Ve vícevrstvých dopředných neuronových sítích, jakými jsou např. Kohonenovy mapy, se signál šíří jedním směrem – dopředu. Existují ale i modely, jako např. Hopfieldova síť, dvousměrná asociativní paměť, nebo Adaptivní rezonanční teorie, ve kterých se signál šíří dvěmi a více směry.

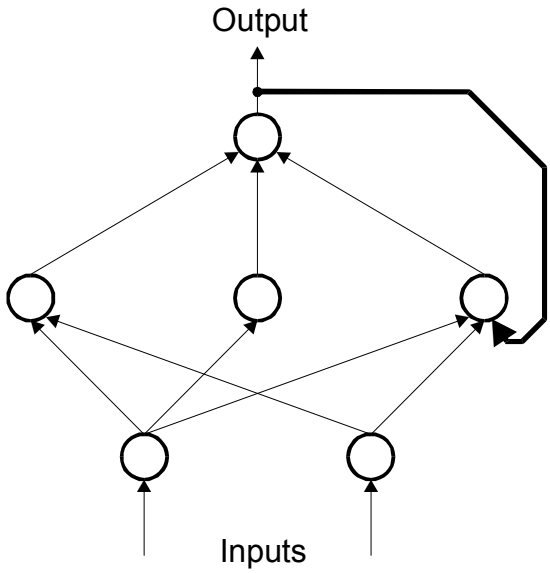
Více informací o nejpoužívanějších typech neuronových sítí lze nalézt ve velmi srozumitelné formě přehledového charakteru např. v [Šnorek, 1996], [Jiřina, 1995], [Blum, 1992] a nebo v touto problematikou hlouběji se zabývajících zdrojích [Freeman, 1991], [Haykin, 1999]. Na závěr si uvedeme příklady alespoň některých základních topologií neuronových sítí, a to jednovrstvou (obr. 7), vícevrstvou dopřednou NS (obr. 8), rekurentní NS (obr. 9) a plošnou (obr. 10).



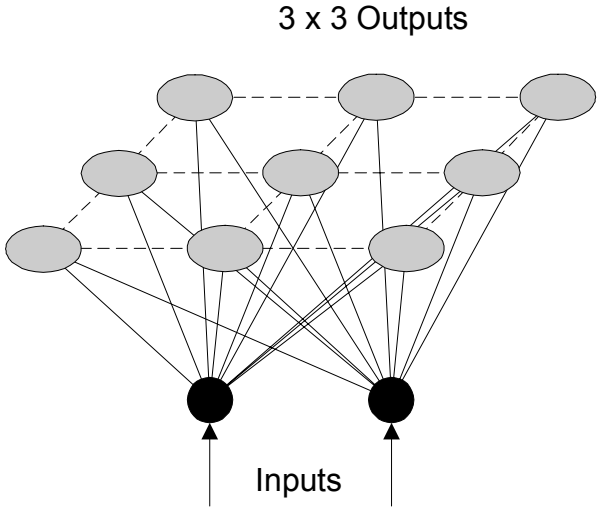
obr. 7 - Hopfieldova neuronová síť



obr. 8 - Vícevrstvá dopředná neuronová síť



obr. 9 - Rekurentní neuronová síť



obr. 10 - Kohonenova síť

4.4 Učení a vybavování

Neuronové sítě pracují většinou ve dvou fázích, a to adaptivní a aktivní. Během adaptivní fáze, dochází v síti ke změnám vah jednotlivých vazeb dle algoritmu učení, tedy k učení NS. V aktivní fázi pak tato naučená neuronová síť "pouze" transformuje vstupní signál na signál výstupní.

4.4.1 Adaptivní fáze (učení)

Během fáze učení jsou neuronové síti předkládány prvky tzv. trénovací množiny vzorů. Tato množina musí být dostatečně obsáhlá, jelikož musí obsahovat určitou obecnou informaci o modelovaném procesu a všechny jeho podstatné charakteristiky.

Z této množiny jsou pak náhodně vybírány jednotlivé vzory a prováděna příslušná korekce synaptických vah neuronů, které jsou na začátku fáze učení nastaveny většinou na malé náhodné hodnoty.

Samotné metody učení můžeme rozdělit na dvě základní skupiny, a to *učení s učitelem*, a *učení bez učitele*.

4.4.1.1 Učení s učitelem

Při učení s učitelem musíme mít k dispozici, buďto přímo požadované (správné) odezvy na jednotlivé vzory, a nebo určité vnější kritérium, které nám správnost odezvy NS na předložený vzor nějakým způsobem ohodnotí. Algoritmy učení pak spočívají většinou v tom, že vypočtou vzdálenost výstupu NS od výstupu požadovaného a upraví váhy neuronů tak, aby se chyba odezvy sítě snížila.

Ať již jsou tedy samotné algoritmy učení jakkoli rafinované, jejich základem je ve většině případů tzv. Delta pravidlo, užívané pro učení (korekci, nebo chcete-li výpočet lépe vyhovujících vah neuronů) jednoduchého perceptronu:

$$w_i(t+1) = w_i(t) + \eta\left[d(t) - y(t)\right]x_i(t),$$

kde

$w_i(t+1)$	… je nová (upravená) hodnota váhy i-tého vstupu neuronu,
$w_i(t)$	… je aktuální hodnota váhy i-tého vstupu neuronu,
η	… je konstanta rychlosti učení,
$d(t)$	… je požadovaná (správná) hodnota výstupu NS pro předložený vzor,
$y(t)$	… je skutečná (aktuální) hodnota výstupu NS pro předložený vzor,
$x_i(t)$	… je aktuální hodnota i-tého vstupu neuronu.

Nejznámější metodou učení s učitelem je tzv. *Backpropagation* (metoda zpětného šíření chyby). Tato metoda je vlastně zobecněným delta pravidlem pro vícevrstvou perceptronovou síť, nazývanou též „Backpropagation síť“. Tato metoda minimalizuje tzv. chybovou funkci E, která je měřítkem velikosti energie nutné ke snížení chyby NS na nulu, jak je patrné z následujícího vztahu:

$$E = \frac{1}{2}\sum_j (d_j - y_j)^2,$$

kde index j prochází přes všechny neurony výstupní vrstvy NS.

Výsledný algoritmus učení pak upravuje váhy dle *zobecněného delta pravidla*. Jednotlivé váhy jsou upravovány současně pouze v jediné vrstvě, přičemž se postupuje od výstupní vrstvy neuronů NS směrem k vrstvě vstupní, a to dle vztahu:

$$w_{ij}^{S}(t+1) = w_{ij}^{S}(t) + \eta \delta_{j}^{S+1} \mu_{i}^{S}(t),$$

kde index S prochází přes všechny vrstvy sítě (od výstupní směrem ke vstupní), index j přes všechny neurony v dané vrstvě a index i přes všechny neurony ve vrstvě o jednu bližší směrem ke vstupní. Tedy $\mu_{i}^{S}(t)$ jsou hodnoty vstupů neuronu dané vrstvy a δ_{j}^{S} jsou chyby neuronů v dané vrstvě.

Při aplikaci tohoto pravidla je třeba dát pozor na to, že pro výpočet chyb neuronů ve výstupní vrstvě se používá jiného vztahu, než při výpočtu chyb neuronů ve vrstvách ostatních. Tím se zde ale již podrobněji zabývat nebudeme. Podrobné informace k tomuto, a také ostatním algoritmům učení neuronových sítí, jako např. Backpropagation se zahrnutím momentu, tzv. Conjugate gradient method, učení RBF sítě, tzv. Counterpropagation a dalším, lze nalézt např. v [Jiřina, 1995], [Blum, 1992], [Freeman, 1991], [Haykin, 1999], [URL, University of Stuttgart].

4.4.1.2 Učení bez učitele

Při učení bez učitele, tzv. samoorganizaci, se žádné vnější hodnocení odpovědi neuronové sítě nevyužívá. Algoritmus učení se naopak snaží ve vstupních datech nalézt určité závislosti bez jakékoli doplňující informace. Základem většiny algoritmů tohoto druhu učení je tzv. Hebbovské učení, které upravuje váhy dle vztahu:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\, y_{i}(t)\, y_{j}(t),$$

kde

$y_i(t)$	… je výstup i-tého neuronu jedné vrstvy NS,
$y_j(t)$	… je výstup j-tého neuronu jiné vrstvy NS,
$w_{ij}(t)$	… je aktuální hodnota váhy mezi těmito dvěmi neurony,
$w_{ij}(t+1)$	… je nová (upravená) hodnota váhy mezi těmito dvěmi neurony,
η	… je konstanta rychlosti učení.

Dalšími metodami patřícími k učení bez učitele jsou např. kompetice (Kohonenova síť), Min-Max učení a učení Hopfieldovy NS. Podrobnější informace, jak k zmíněným, tak i k mnoha dalším učícím algoritmům a samotným typům neuronových sítí lze nalézt v odkazech uvedených na konci předchozího odstavce.

4.4.2 Aktivní fáze (vybavování)

V aktivní fázi využíváme naučených schopností neuronové sítě ke zpracování vstupních dat. Po předložení daného vstupu dojde k aktivaci neuronů ve vstupní vrstvě a tedy ke změnám jednotlivých vstupů neuronů ve vrstvě následující. Následná aktivace příslušných neuronů této vrstvy znamená změnu vstupů neuronů vrstvy následující a tak postupně dojde k průchodu informace až na samotné výstupy NS. Obecně mluvíme o vzniku nerovnovážného stavu, po jehož ustálení a opětovném spočinutí neuronové sítě ve stavu rovnovážném, bude na výstupech k dispozici požadovaná odpověď na vstupní data. Ustálení NS není u sítí, s větším počtem směrů šíření informace a především u sítí rekurentních (kde je možný vznik oscilací), zcela triviální záležitostí, a je třeba mu věnovat značnou pozornost.

5 Existující implementace

V této kapitole uvedu několik příkladů prezentujících jednak možnosti využití neuronových sítí, jednak možnosti distribuované umělé inteligence a jejího reaktivního přístupu. Jsou zde uvedeny, resp. stručně nastíněny, projekty z této oblasti, které jsem ve většině případů nalezl na Internetu, a které mně určitým způsobem zaujaly a inspirovaly. Některé projekty jsou uvedeny pouze pro demonstraci možností dosažitelných pomocí daného přístupu. Tyto projekty mne spíše "jen" směrovaly k oblastem, k nimž jsem dále soustředil svoji pozornost (především oblast neuronových sítí), a jsou tedy popsány jen zevrubně. Projektům, které mne určitým způsobem inspirovaly a z nichž jsem využil některé myšlenky, je samozřejmě věnována pozornost větší. Nicméně i tyto jsou z pochopitelných důvodů uvedeny pouze přehledově. Podrobnější informace lze nalézt v příslušných odkazech.

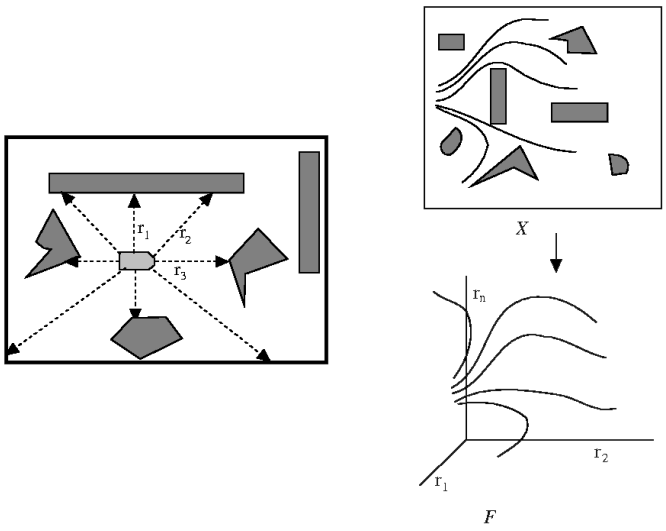
5.1 Lokalizace v prostoru senzorů

název projektu: ***Environment learning and localization in "sensor space"***
autor: Ben Kröse
zdroj: Proc. of the Tenth Netherlands/Belgium Conf. on Artificial Intelligence, Listopad 1998, str. 229-239.

Pro navigaci robota je potřeba globální informace o prostředí, ve kterém se robot nachází. Na rozdíl od standardního přístupu, kdy je tato informace reprezentována mapou prostředí, která udává polohu objektů a volný prostor, je v tomto projektu popsán zajímavý přístup, u něhož globální informaci nereprezentuje geometrický model prostředí, ale model všech senzorických dat robota.

Tedy poloha robota není určena souřadnicí v prostoru, ale souřadnicí v prostoru senzorických dat. V tomto vícedimenzionálním prostoru, jehož dimenze je určená počtem senzorů robota, je reprezentován model prostředí a provádí se v něm samotné plánování cesty a detekce překážek robota.

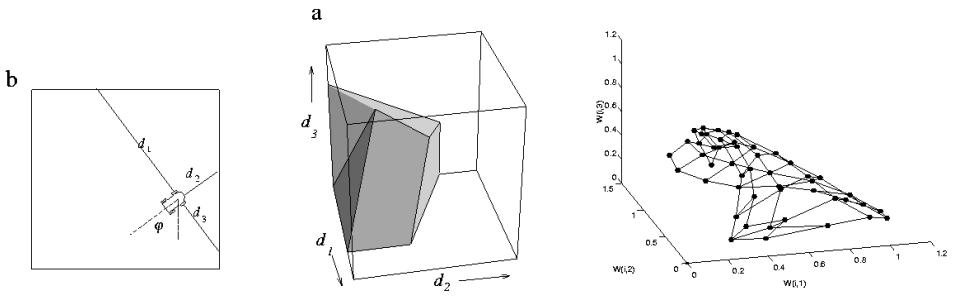
Prostor senzorů je tedy definován jako prostor naměřených hodnot poskytnutých přímo výstupy senzorů, popř. z nich určitým způsobem odvozených. Tento prostor je obecně vícedimenzionální, nicméně počet stupňů volnosti dat v něm reprezentovaných je jen 3. To je dáno stupni volnosti robota, které jsou 3 (translace X,Y + rotace, popř. bez rotace jen 2). Z toho vyplývá, že všechna data budou ležet na 3 (popř.2) rozměrných křivkách, viz obr. 11. Takovýto model pak tedy lze použít pro vlastní navigaci. Přitom je tato reprezentace shodná s reprezentací využívanou při reaktivním přístupu.



obr. 11 - Trajektorie v prostoru a prostoru senzorických dat

Pro implementaci tohoto modelu je využita samoorganizující neuronová síť (Kohonenova NS). Tato NS hledá ve vstupních datech podobnosti a neurony se snaží uspořádat tak, aby byla schopna množinu dat klasifikovat do tolika skupin, kolik je počet neuronů v síti. V tomto případě se tedy NS naučí přiřadit příslušné vektory vstupů (vektory hodnot senzorů) jednotlivým neuronům, a tudíž je pak schopna transformovat prostor do prostoru dat. Jelikož je skutečná rozměrnost (dimenze) dat rovna 3 je použita NS s uspořádáním neuronů do trojrozměrné matice. Bližší informace o použitém algoritmu viz [Kröse, 1998].

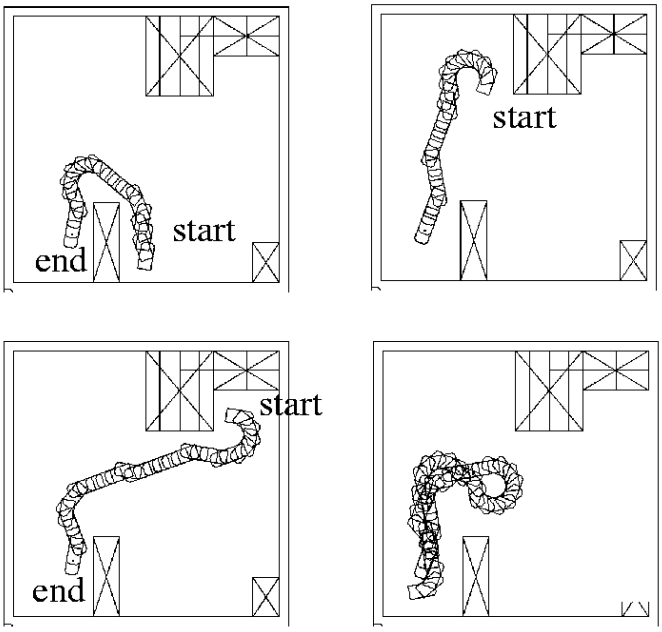
Pro lepší představu je uveden obr. 12, na kterém je uveden zjednodušený případ pro robota se dvěma stupni volnosti a třemi senzory. Zcela vpravo je 2D Kohonenova síť trénovaná na těchto datech. V samotném projektu je však použit mnohem složitější model s 16 senzory a třemi stupni volnosti robota.



obr. 12 - Transformace prostoru do prostoru dat a odpovídající NS

Výsledný model prostředí je tedy reprezentován diskrétními pozicemi v prostoru, jimž odpovídají jednotlivé neurony v NS. Jelikož však sousední pozice neuronů neznamená, že existuje také přímý přechod mezi jimi reprezentovanými stavy v prostředí, je plánování převedeno na Markovský rozhodovací problém, jehož popis je v [Kröse, 1998] a zde se jím zabývat nebudeme.

Jelikož naplánovaná cesta je definovaná v prostoru senzorických dat, lze pro navigaci robota použít reaktivní řídicí systém. Na obr. 13 jsou uvedeny příklady výsledných trajektorií pohybu robota pro pozici cíle v levém dolním rohu místnosti a čtyři různé výchozí polohy robota.



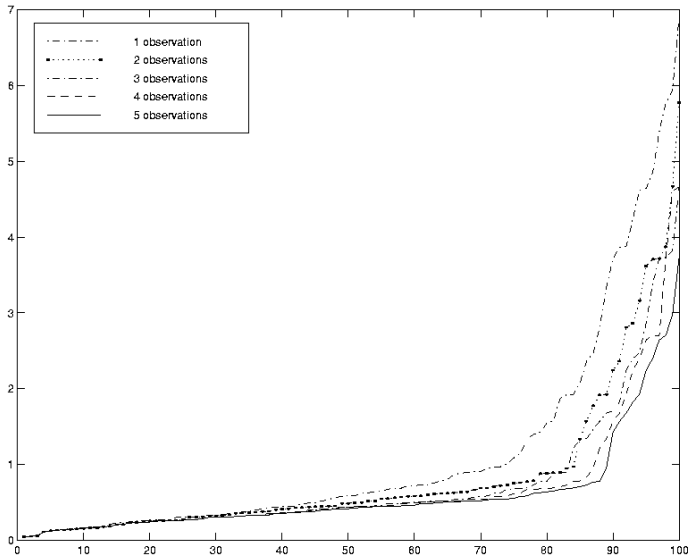
obr. 13 - Trajektorie pohybu robota

Díky konečnému rozsahu použitých senzorů, a především díky jejich specifickému zaměření na určitou veličinu, však nastává problém, kdy mnoho pozic v prostředí má stejnou geometrickou strukturu, díky níž nejsme pomocí těchto senzorů schopni tyto pozice navzájem rozlišit. Proto dál projekt pokračuje směrem k použití vizuálního systému, tedy k získávání mnohem většího obsahu dat pomocí kamery.

Hlavní myšlenkou je to, že skutečná rozměrnost (dimenze) informace z obrázků pořízených kamerou omezena stupni volnosti kamery a podmínkami osvětlení. To, že naměřená data budou ležet na 3 rozměrných křivkách, se nezmění. Pouze tyto křivky budou ležet ve vícedimenzionálním prostoru obrázků. To jakým způsobem vytváření modelu prostředí a lokalizace polohy v něm v tomto případě probíhá je podrobně uvedeno v [Kröse, 1998].

V experimentu s vizuální informací z prostředí, bylo nejprve v simulačním systému "OASIS" vytvořeno prostředí laboratoře a nasnímány pohledy z 936-ti různých poloh. Rozlišení obrázků bylo 64x40 pixelů. Na základě těchto dat byla vytvořena reprezentace prostředí laboratoře. Poté bylo náhodně vygenerováno 100 různých pozic a orientací robota a vygenerován obrázek pořízený kamerou při každé z těchto poloh.

Výsledkem bylo to, že pro více než 70 poloh byla chyba určení polohy menší než 0.7m, jak je naznačeno na obr. 14, kde je uvedena chyba určení polohy pro pět různých množin 100 náhodně vygenerovaných poloh.



obr. 14 - Chyba určení polohy při použití kamery

5.2 Shromažďování potravy

název projektu: ***Learning the foraging task in o multi-robot domain***
autor: Monica Nicolescu
zdroj: www-robotics.usc.edu/~monica, University of Southern California, 1998

V tomto projektu je využito tzv. *Q-Learning* algoritmu učení, který patří do oblasti nazývané *Reinforcement Learning*, tedy posilovaného učení. *Reinforcement Learning* patří mezi učení s učitelem, který dodává pouze informaci o vhodnosti (odměna, trest) akce provedené v minulém kroku.

Projekt předkládá pojetí posilovaného učení umožňujícího učení v dynamickém prostředí více spolupůsobících robotů. Doba učení, která bývá většinou neúměrně velká, je zmenšena použitím tzv. "Behavior-Based Architecture", která využívá akčních primitiv a podmínek jejich aktivace k odpoutání se od detailů nejnižší (low-level) řídící úrovně, a tím snižuje rozsáhlost stavového prostoru. K učení vhodných reakcí je použito různých hodnotících (reinforcement) funkcí a tzv. "progress estimator" [Matarić, 1997], což je vlastně rozšíření hodnotícího signálu o průběžné hodnocení směru pohybu robota.

Příkladem, na kterém je demonstrován především přínos průběžného hodnocení (progress estimator), je úloha shánění potravy skupinou robotů. Tito musí nalézt efektivní strategii svého jednání, tedy nalézt správné přiřazení akčních primitiv k příslušným podmínkám prostředí (stavům senzorů).

Jak jsem již uvedl, je v projektu využito tzv. *Q-Learning* algoritmu učení. Ten spočívá v tom, že k rozhodnutí do kterého z možných následujících stavů stavového prostoru se ze současného stavu přejde, tedy která akce z množiny akčních primitiv

bude provedena, se využívá tzv. „Q – Utility Function“ tedy funkce určující užitečnost či taktičnost provedení dané akce v příslušném stavu prostředí. Učící algoritmus se snaží tuto funkci přiblížit k optimální strategii, tj. aby zvolena byla ta akce, která bude v budoucnu znamenat největší přínos (odměna). Podrobné informace o samotném algoritmu lze nalézt např. v [Matarić, 1997], [Sheppard, 1995].

Každý robot má tedy danou množinu základních akcí (akčních primitiv):

- *Forrage* - náhodný pohyb robota s vyhýbáním se překážkám
- *Grasp* - zvednutí potravy
- *Home* - přesun mobota do stanoveného místa (základna)
- *Drop* - položení potravy

Jediným vestavěným mechanizmem je vyhýbaní se překážkám, které má prioritu před všemi ostatními činnostmi robota.

Množina podmínek (stavů senzorů) obsahuje:

- *has-food* - je pravdivá (true) když robot drží potravu
- *at-home* - je pravdivá (true) když se robot nachází na základně
- *sense-food* - je pravdivá (true) když robot detekuje v blízkosti nějaké jídlo

Podmínky aplikovatelnosti základních akcí jsou:

- *Forrage* - vždy
- *Grasp* - jestliže *sense-food* = true
- *Home* - vždy
- *Drop* - jestliže *has-food* = true

Úlohou robotů je tedy přiřadit k možným kombinacím podmínek (možným stavům prostředí) spuštění vhodných akčních primitiv. Při vlastním učení tedy algoritmus upravuje Q – funkci, která přiřazuje daným podmínkám spuštění příslušných akcí, dle hodnotícího (reinforcement) signálu z prostředí (odměna, trest).

Události vyvolávající okamžitou odměnu jsou:

- grasped-food
- dropped-food-at-home

Událostí vyvolávající okamžitý trest je pouze:

- dropped-food-away-from-home

Hodnotící (reinforcement) funkce tedy je:

$$R(c) = R_B(c) + R_H(c),$$

kde

$$R_B(c) = \begin{cases} pos\_reward\_grasp & if \quad grasped - food \\ pos\_reward\_drop & if \quad dropped - food - at - home \\ neg\_reward\_drop & if \quad dropped - food - away - from - home \\ 0 & otherwise \end{cases},$$

$R_H(c)$ … "progress estimator" – je aktivní pouze když robot drží potravu, a dává kladnou hodnotu při přibližování se k základně a negativní hodnotu při vzdalování se od základny.

Byly provedeny experimenty ukazující schopnost algoritmu přiblížit se k správným relacím mezi podmínkami a akcemi, viz tab. 2. Experimenty byly provedeny pro odlišné hodnotící signály a sledován průběh učení, tak aby bylo možno ukázat vliv hodnotícího signálu na rychlost učení.

STAV	**AKCE**
has-food & *at-home*	Drop
no-food & *at-home*	Forrage
sense-food & *at-home*	Forrage
has-food & *not-at-home*	Home
no-food & *not-at-home*	Forrage
sense-food & *not-at-home*	Grasp

tab. 2 – Správné relace mezi stavy a akcemi

Experiment 1:

$$R_B(c) = \begin{cases} pos\_reward\_drop & if \quad dropped - food - at - home \\ neg\_reward\_drop & if \quad dropped - food - away - from - home \\ 0 & otherwise \end{cases}$$

Experiment 2:

$$R_B(c) = \begin{cases} pos\_reward\_grasp & if \quad grasped - food \\ pos\_reward\_drop & if \quad dropped - food - at - home \\ neg\_reward\_drop & if \quad dropped - food - away - from - home \\ 0 & otherwise \end{cases}$$

Experiment 3:

Hodnotící signál z experimentu 2 byl rozšířen o již zmíněný "progress estimator".

V tab. 3 jsou uvedeny výsledky experimentů provedených s jediným robotem přítomným v prostředí. V tab. 4 jsou pak uvedeny výsledky získané při současném učení pěti robotů. Výsledky jsou uvedeny v procentech maximální úspěšnosti (tj. přiřazení relací dle tab. 2). Jednotlivé sloupce tabulek odpovídají stavům po příslušném počtu kroků učení.

	40	**500**	**1000**	**3000**	**5000**	**END**
Experiment 1	0	0	33.3	66.6	83.3	83.3
Experiment 2	0	16.6	66.6	83.3	83.3	83.3
Experiment 3	0	50	83.3	83.3	83.3	83.3

tab. 3 - Výsledky experimentů s jedním robotem v prostředí

	40	**500**	**1000**	**3000**	**5000**	**END**
Experiment 1	0	0	0	33.3	53.3	53.3
Experiment 2	0	16.6	36.6	76.6	76.6	76.6
Experiment 3	0	16.6	70	76.6	76.6	76.6

tab. 4 - Výsledky experimentů s pěti roboty v prostředí

Je tedy vidět že čím více informace, ve formě hodnotícího signálu, robotům zprostředkujeme, tím rychleji se naučí potřebné relace mezi stavy a akcemi a tím rychleji dosáhnou stanovené úrovně znalostí. Z třetího experimentu pak vyplývá, že dodáním určité obecné znalosti určité oblasti, ve formě tzv. "progress estimator" (průběžný odhad), výrazně rychlost učení zvýšíme. Z tab. 4 vidíme, že přítomnost více robotů v prostředí, díky jejich vzájemným střetům a překážení, konvergenci učení sníží. I zde je ale vidět obrovský přínos implicitní znalosti určité oblasti.

5.3 Behaviour Synthesis Architecture

název projektu: ***Many hands make light work? An investigation into behaviourally controlled co-operant autonomous mobile robots***
autor: D. P. Barnes, R. A. Ghanea-Hercock, R. S. Aylett , A. M. Coddington
zdroj: Proceedings of the first International Conference on Autonomous Agents, February 5-8 (1997), str. 413-420

Charakteristikou tohoto výzkumu je opět reaktivní přístup řízení. Základem je implementace řídící struktury umožňující autonomní řízení dvou kooperujících robotů, jejíž základ byl rozvinut v předcházejícím projektu MACTA (Multiple Automata for Complex Task Achievement, Barnes & Aylett, 1994), a která je známá jako *Behaviour Synthesis Architecture* (BSA). Pomocí této architektury byla realizována úloha

přemisťování předmětu dvěma kooperujícími mobilními roboty, při současném vyhýbání se překážkám, a tedy ukázáno, že reaktivní řízení je v takovýchto úlohách efektivním prostředkem.

V tomto projektu je soustředěna pozornost především na výrazně rostoucí komplexnost požadovaných úloh. Takovouto úlohou je například kooperativní demontáž, která je inspirována problémem vyřazení zastaralých jaderných elektráren z provozu. Vývoj dálkově ovládaných zařízení v této oblasti je v současné době ještě v plenkách. Problémem je, že demontážní úlohy vyžadují použití mnoha robotů současně, což znamená velký počet stupňů volnosti systému a klade tak na jeho řízení obrovské nároky.

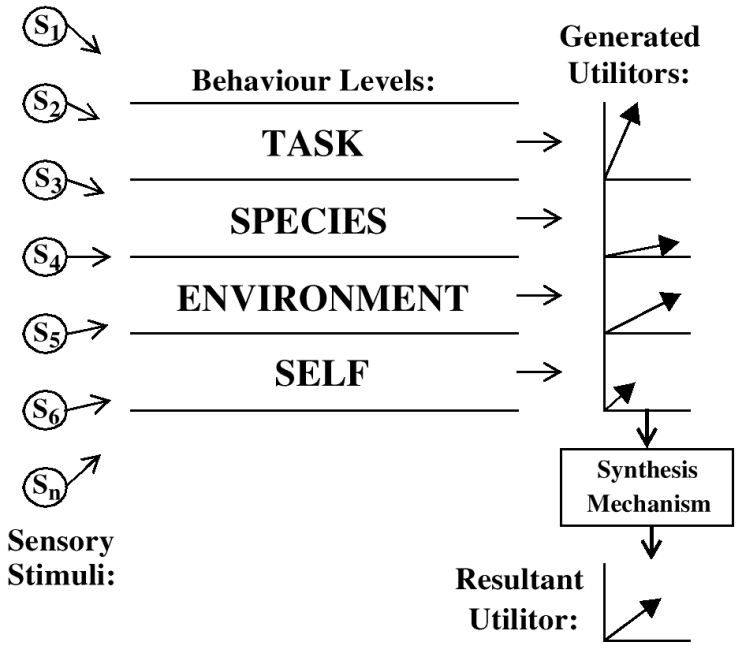
Demonstrační úlohou může být předání objektu mezi dvěma roboty (Fred a Ginger) a nikoliv "jen" jeho společná přeprava. Tato úloha je popsána následujícím způsobem:

1. Fred jede přes laboratoř do doku. Ginger ho následuje.
2. Fred vjede do doku a uchopí předmět (část trubky).
3. S předmětem se Fred začne pohybovat směrem k cílové pozici. Ginger jede blízko něj.
4. Fred předá za jízdy předmět Ginger a ta ho dopraví do cílové pozice.

Je jasné, že jde o komplexní problém zahrnující navigaci, vyhýbání se překážkám, zajetí do doku, sledování cíle, uchopení a předání předmětu, a společnou přepravu předmětu. Řešení takto komplexní úlohy skrývá mnoho problémů. Tím prvním je požadované množství způsobů jednání (reakcí) robota, což může v daném časovém okamžiku znamenat soupeření mnoha z nich. Jelikož je robot schopen v daném čase vykonávat pouze jedinou akci, je potřeba nějakým způsobem tyto konflikty řešit. Jelikož úloha vyžaduje velké množství vzájemně se ovlivňujících způsobů jednání, je jejich ruční rozvržení takovým způsobem, aby bylo dosaženo požadovaného chování, prakticky nemožné a je tedy třeba nějakého mechanismu jejich automatické adaptace. Posledním, ale nejpodstatnějším problémem je, že standardní reakční architektura je navržena na vykonávání jednotlivých akcí, a ne na vykonávání sekvencí těchto akcí. Tedy, pro přizpůsobení se požadavkům schopnosti robota vykonávat sekvence akcí, je potřeba nějaký mechanismus jejich plánování.

Pro vyřešení vzájemných konfliktů jednotlivých akcí, jakým je např. na jedné straně snaha robota zůstat v dostatečné vzdálenosti od všech ostatních objektů prostředí (včetně jiných robotů) a na druhé straně snaha držet se v těsné blízkosti robota s nímž kooperuje, je použita již zmíněná BSA viz obr. 15. V této architektuře jsou rozlišeny čtyři vrstvy:

- SELF - soustředí se na maximalizaci a naplnění interních zdrojů (např. zůstat stát z důvodu ušetření energie baterie)
- ENVIRONMENT - obsahuje akce související s ostatními objekty v prostředí (např. vyhýbání se překážkám)
- SPECIES - zahrnuje akce související s kooperací robotů (např. udržování správné pozice vůči kooperujícímu robotovi)
- TASK - obsahuje akce specifické pro speciální úlohy (např. dosažení dané polohy - dok s předmětem)



obr. 15 - Behaviour Synthesis Architecture (BSA)

Podněty senzorů poskytují jednotlivým úrovním jednání robota odpovídající informace jak o stavu prostředí, tak i o vnitřním stavu robota. V každé úrovni je pak vygenerován příslušný požadavek na pohyb, vztahující se k požadovanému řízení. Každá úroveň může obsahovat několik akčních vzorů, tzv. *behaviour patterns*, skládajících se z požadované pohybové reakce a k ní asociované důležitosti. Je jasné, že jak požadovaná pohybová reakce, tak její důležitost, jsou funkcemi (f_r, f_u) podnětů senzorů.

$$\mathbf{bp} = \begin{bmatrix} r \\ u \end{bmatrix} \qquad \text{a} \qquad \begin{aligned} r &= f_r(s) \\ u &= f_u(s) \end{aligned},$$

kde

bp … je akční vzor (*behaviour pattern*),

r … je požadovaná pohybová akce,

u … je důležitost či užitečnost této akce,

s … je podnět senzorů.

Pomocí takto zavedených r, u můžeme již snadno definovat vektor, známý jako *utilitor*, jehož velikost je rovna důležitosti u a úhel je úměrný požadavku pohybu r. Pokud v daném okamžiku t dojde k vygenerování konfliktních požadavků pohybu,

je jednoduše lineární superpozicí relevantních *utilitorů* (translace, rotace apod.) vypočten výsledný *utilitor* a z něj pak výsledná důležitost a výsledný požadavek pohybu:

$$\mathbf{UX}_t = \sum_{n=1}^{m} u_{(t,n)} e^{j r_{(t,n)}} \quad , \qquad uX_t = \frac{|UX_t|}{m} \quad \text{a} \qquad rX_t = \arg(\mathbf{UX}_t) ,$$

kde

$\mathbf{UX}_t$	… je výsledný *utilitor*,
$u_{(t,n)} e^{j r_{(t,n)}}$	… jeden z relevantních *utilitorů* v okamžiku t,
m	… počet relevantních *utilitorů* v okamžiku t,
uX_t	… je výsledná důležitost,
rX_t	… je výsledný požadavek pohybu.

K dříve zmíněné požadované automatické adaptaci bylo použito fuzzy logiky, s reprezentací pravidel pomocí FAM (fuzzy associative memory matrix). Jak víme, BSA používá f_u (*utility function*) pro každý akční vzor (*behaviour pattern*). Právě tato funkce umožňuje přímý způsob adaptace relativní užitečnosti každého způsobu jednání z vyšší vrstvy řízení.

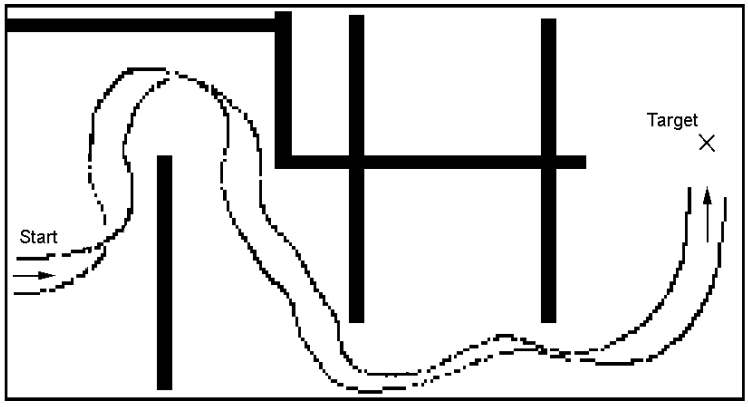
Navržená vrstva fuzzy řízení přijímá přímé vstupy senzorů a aplikuje zápornou zpětnou vazbu na užitečnost aktivovaných akcí. Příspěvek každé akce je tudíž dynamicky přizpůsobován současnému stavu podnětů z prostředí.

Velkou výhodou je, že užitečnost mnoha konfliktních akcí může být dynamicky upravována místo generováním výsledných *utilitorů*, rozšířením počtu vstupně výstupní dimenze báze pravidel, tedy bez modifikace samotných akčních vzorů. Jelikož přitom ale rapidně narůstá počet možných kombinací jednotlivých pravidel je k jejich rozvržení používáno standardních metod genetických algoritmů, blíže viz [Barnes, 1997].

Akční vzory se mohou navíc ovlivňovat i způsobem, který pro vlastního robota není potřebný. I když funkce užitečnosti jsou ideální pro generování výsledného jednání robota, jsou závislé na senzorech a ne na úlohách a podúlohách. Může nastat situace, kdy je potřeba užitečnost daného akčního vzoru vynuceně snížit na nulu, bez ohledu na hodnotu vstupu.

Schůdným řešením by bylo umožnit na základě struktury úlohy vytvoření kontextů, umožňujících povolení aktivace pouze určité části akčních vzorů. Pro tento účel byl v BSA vyvinut scénář, tzv. *behaviour script,* skládající se z paketů, z nichž každý obsahuje triplet [předcházející podmínky, akční vzor, následné podmínky]. Jakmile je daný paket vyřízen, jsou nastaveny podmínky pro provedení dalšího paketu, a tak pořád, dokud není proveden celý scénář. Tento proces představuje ideální mechanismus plánování akcí.

Rozšířená architektura BSA, popsaná v tomto článku byla úspěšně odzkoušena na úloze uvedené v úvodní části textu. Dráha obou spolupracujících robotů je uvedena na obr. 16.



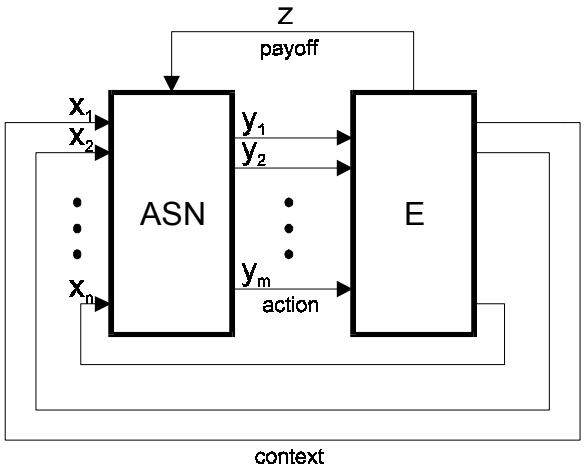
obr. 16 - Dráha spolupracujících robotů užívajících BSA s adaptací

5.4 Využití orientačních majáků

název projektu: ***Landmark Learning: An Illustration of Associative Search***
autor: Andrew G. Barto, Richard S. Sutton
zdroj: Biological Cybernetics, Springer Verlag, 1981,vol. 42, str. 1-8

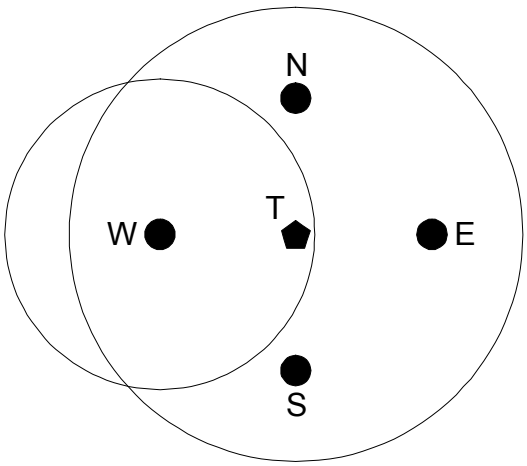
V tomto článku je prezentováno získávání a využívání prostorových znalostí na principu asociativního vyhledávání. Základem je tzv. *Associative Search Network* (ASN), tedy asociativně vyhledávací neuronová síť, která spadá mezi neuronové sítě s posilovaným učením (*Reinforcement Learning*). Tato NS hledá pro každý vstupní vzor (vektor vstupů) takový výstupní vzor (akci), který určitým způsobem optimalizuje externí hodnotící (reinforcement) signál. Na příkladu, kdy ASN řídí pohyb v prostředí obsahujícím různé zdroje vůní, jsou názorně předvedeny vyhledávací, asociativní a především zobecňující vlastnosti ASN.

Princip interakce ASN z prostředí je patrný z obr. 17. V každém okamžiku má ASN k dispozici informaci o stavu prostředí ($x_1(t), x_2(t), \ldots, x_n(t)$) a hodnotící signál $z(t)$ (tzv. "payoff"). ASN tedy na základě stavu prostředí (E - environment) resp. svých senzorů, vygeneruje akci ($y_1(t), y_2(t), \ldots, y_m(t)$) a po jejím provedení, tzn. o jeden krok později, dostane ohodnocení provedení této akce v dané situaci (při daném stavu senzorů), a to prostřednictvím *payoff* signálu. ASN se pak snaží pro každý vstupní vektor (neboli kontext) nalézt akci, která tento hodnotící signál maximalizuje. Jinými slovy, snaží se pro každou situaci nalézt nejlepší či nejvýhodnější akci. Podrobněji se této problematice věnuje [Barto(2), 1981].



obr. 17 - Interakce ASN a prostředí (E)

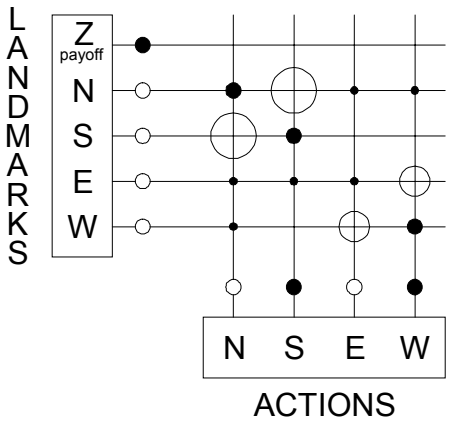
Pokud je na *Associative Search Network* pohlíženo jako na systém řídící pohyb v prostředí, je vstupní informace (kontext) určitým způsobem spojena s pozicí v prostředí a výstupní akce s následným pohybem. Pro ilustraci bylo vytvořeno prostředí obsahující kromě cíle ještě čtyři orientační majáky (zdroje různých vůní). Prostředí je naznačeno na obr. 18. Jednotlivé majáky (N, S, E, W) a také cílový bod (T) jsou charakteristické svojí vůní, kterou je možno zaznamenat ("ucítit") do určité vzdálenosti. Dosahy vůně cíle a vůně jednoho z majáků jsou na obrázku znázorněny kružnicemi. Intenzita dané vůně se vždy směrem od jejího zdroje lineárně snižuje a za hranicemi dosahu je nulová.



obr. 18 - Prostředí s orientačními majáky

Vstup ASN je tedy určen pozicí v prostoru, tj. intenzitami zaznamenaných vůní jednotlivých majáků. Vůně cíle je pak brána přímo jako hodnota *payoff* signálu. *Associative Search Network* bude tedy mít celkem pět vstupů (včetně hodnotícího *payoff* signálu).

Jedno z možných znázornění ASN je uvedeno na obr. 19. Vlevo jsou vertikálně seřazeny vstupy (payoff,N,S,E,W), jejichž hodnoty odpovídají intenzitám zaznamenaných vůní. Dole jsou horizontálně uvedeny výstupy, jenž odpovídají požadavkům pohybu ve směru světových stran. Průsečíky horizontálních a vertikálních čar pak reprezentují váhy neuronů ASN. Kružnice na průsečících odpovídají svým poloměrem velikostem příslušných vah, přičemž plné kružnice znamenají záporné hodnoty a prázdné kladné hodnoty. Vstup *payoff* signálu samozřejmě žádné váhy přiřazeny nemá, jelikož je využíván jako hodnotící signál a na obrázku je uveden jen pro úplnost.



obr. 19 - ASN řídící pohyb v prostředí

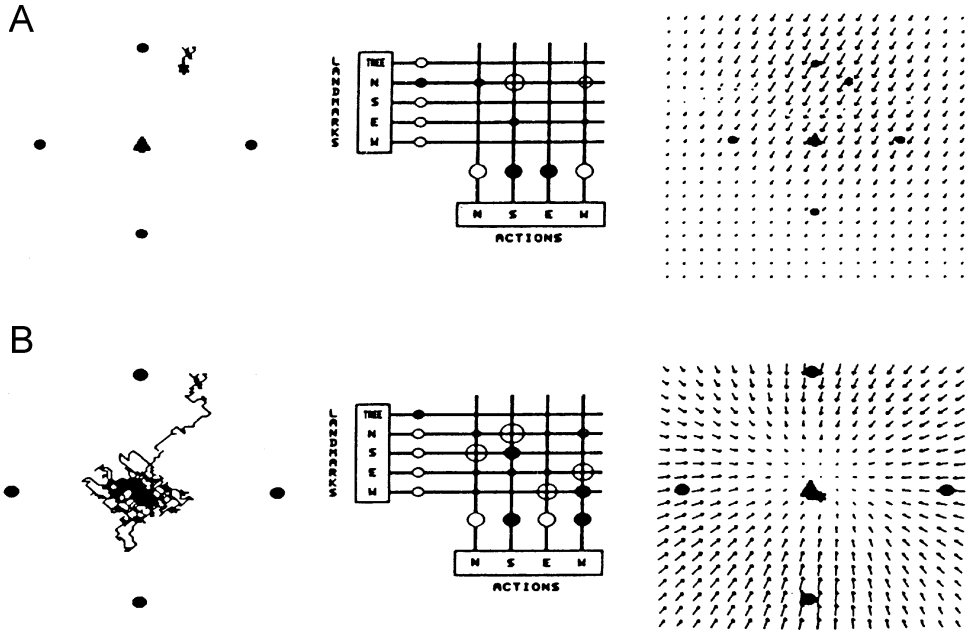
Jak již bylo řečeno, ASN se snaží hodnotící signál *payoff* maximalizovat, a jelikož je při této interpretaci hodnotícím signálem přímo samotná vůně cíle (T), bude výsledkem pohyb směrem k tomuto cíli. Po naučení, by tedy jednotlivé váhy v síti měly být nastaveny takovým způsobem, aby pohyb směrem k cíli podporovaly.

Na obr. 19 je uvedena již částečně naučená síť, což je z odpovídajících hodnot vah neuronů patrné. Například pro vstup intenzity vůně majáku N je vidět, že jsou přiřazeny malé záporné hodnoty pohybům do světových stran N, E a W a výrazná kladná hodnota pro pohyb S (tedy na jih). Pro pozici v blízkosti majáku N, bude tedy výrazně preferován pohyb směrem k S a naopak potlačovány pohyby N, E a W. A to je jak vidíme přesně to co potřebujeme, jelikož při dodržení těchto pravidel bude prováděný pohyb směřovat k cílovému bodu (T). Podobně jsou nastaveny i váhy pro ostatní stavy vstupů.

ASN se tedy snaží najít cíl sledováním diferencí signálu jeho vůně a současně si při tom k jednotlivým kombinacím vstupů asociuje takové akce, které budou mít za následek právě pohyb směrem k tomuto cíli.

Nejdůležitější je si uvědomit, že jako vstupy nefigurují přímo souřadnice pozice v prostoru, ale pouze vzdálenost této pozice od jednotlivých majáků. Naučená ASN je tím pádem schopna generovat efektivní akce i pro pozice, ve kterých se nikdy přímo nenacházela a tato její zobecňující schopnost je zdrojem obrovských možností.

Na obr. 20 je schopnost generalizace patrná z pole vektorů směru, kterým by se ASN v jednotlivých pozicích prostředí vydala, uvedeného zcela vpravo. Uprostřed obrázku je uvedeno nastavení jednotlivých vah ASN a vlevo její trajektorie pohybu. Pro znázornění průběhu učení jsou uvedeny stavy po 35 (viz A) a 800 (viz B) krocích.



obr. 20 - Trajektorie pohybu pro ASN s využitím orientačních majáků

A – trajektorie 35 kroků pohybu ASN je zcela vlevo, uprostřed je uvedeno nastavení synaptických vah neuronů sítě po těchto 35 krocích (kružnice na průsečících odpovídají svým poloměrem velikostem příslušných vah, přičemž plné kružnice znamenají záporné hodnoty a prázdné kladné hodnoty) a zcela vpravo je pole vektorů směru, kterým by se ASN v jednotlivých pozicích prostředí na základě svých dosavadních znalostí vydala. B – trajektorie pohybu, nastavení synaptických vah a pole vektorů počátečního směru pohybu po 800 krocích. Z obou polí vektorů počátečního pohybu jsou zcela patrné generalizující schopnosti ASN, jelikož tyto vektory jsou orientovány správným směrem i v místech, kde se ASN v průběhu svého pohybu nikdy nenacházela.

6 Vlastní řešení

Zadáním této práce bylo navrhnout řídící systém mobilního robota (mobota), umožňující koordinaci jeho jednání s jiným mobotem, popřípadě v rámci skupiny. Dále pak navrhnout aplikační úlohu demonstrující možnosti takovéhoto řídícího systému a konečně navrhnout a realizovat (naprogramovat) prostředí, umožňující provést potřebné simulace na počítači. Jak při návrhu aplikační úlohy, tak při návrhu řídícího systému jsem se v maximální míře soustředil na umožnění pozdější realizace pomocí již existujících mobotů, jimiž disponuje mobotická skupina na katedře kybernetiky elektrotechnické fakulty ČVUT v Praze. Z důvodu časové náročnosti jejich nutného doplnění o některé další zdroje informace, jsem však, na pokyn vedoucího práce, od oblasti implementace úlohy na reálných robotech ustoupil. Zmíněné oblasti a stejně tak dalšímu vývoji samotného řídícího systému bych se chtěl věnovat v rámci svého doktorandského studia.

Po prostudování základních principů chování živých organismů z pohledu etologie samotné a základních přístupů a metod distribuované umělé inteligence, jsem se rozhodl pro reaktivní, chováním motivovaný řídící systém, a využití neuronové sítě pro realizaci vrstvy koordinace (pojem vrstvy bude objasněn v odstavci 6.1 - *Řídící systém mobota*). Důvody, které mne k tomuto rozhodnutí vedly, jsem, snad dostatečným způsobem, objasnil v předcházející části této práce, a to především v kapitolách 1 - *Úvod* a 3 - *Výběr použitého přístupu*.

V následujících odstavcích tedy uvedu informace o postupu mé práce a dosažených výsledcích. Nejprve se budu věnovat struktuře vlastního řídícího systému, poté návrhu demonstrační úlohy a specifikaci požadavků, které bude tato klást na implementaci simulátoru, dále pak realizaci prostředí simulátoru a nakonec dosaženým výsledkům.

V průběhu mé práce samozřejmě docházelo k neustálému formování mého pohledu na danou problematiku díky narážení na problémy nejrůznějšího charakteru, které bylo potřeba nějakým způsobem řešit. To se v mnoha případech podařilo pouze úpravou na úrovni vlastního algoritmu, případně dotvořením dalších pomocných prostředků apod. Nicméně nastaly i situace, které určitým způsobem změnily můj celkový pohled na zvolený přístup k řešení dané oblasti a posléze k jeho přepracování. I tyto situace byly samozřejmě obrovským přínosem, neboť mne tak alespoň vyvedly z omylu. Některé závažnější zlomy v průběhu vývoje a simulace samozřejmě uvedu v příslušných odstavcích, přílišnými podrobnostmi o úpravách zejména na úrovni implementace algoritmu však tuto práci zbytečně zatěžovat nebudu.

Chci jen zdůraznit, že struktura řídícího systému, způsob její implementace v prostředí simulátoru a taktéž konstrukce simulátoru samotného, nebyly od počátku navrženy přímo v takové formě, jak jsou zde nyní podány, ale jsou výsledkem dynamického vývoje.

6.1 Řídící systém mobota

V tomto odstavci, se nejprve budu věnovat navrhnuté globální struktuře řídícího systému mobota. Po té se podrobněji zaměřím na tu část, která je z devadesáti procent náplní této práce, a sice podrobnému návrhu jedné z možných implementací jeho vrstev.

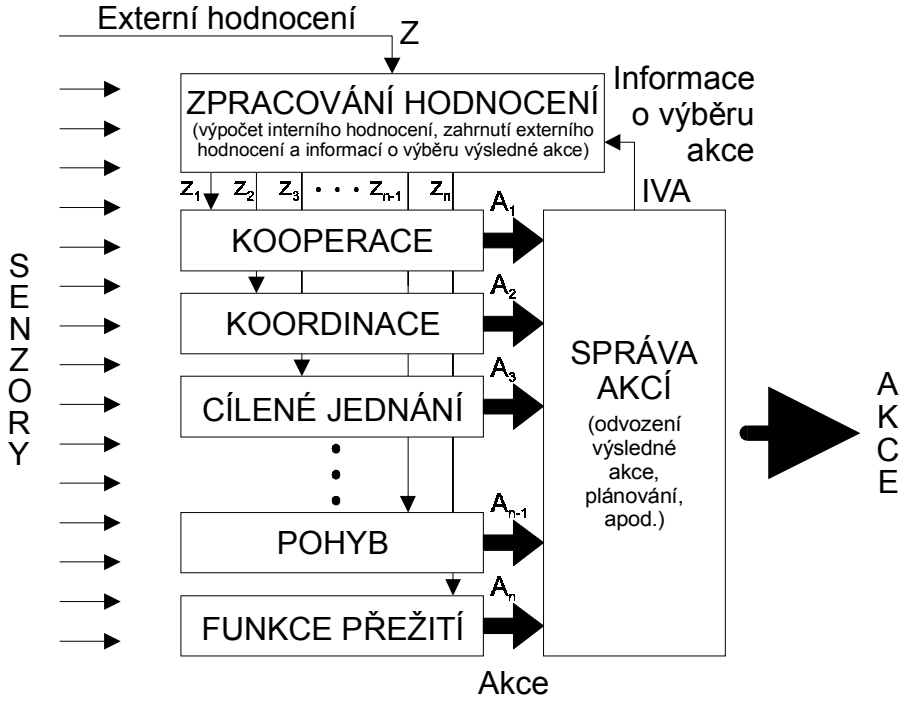
Způsob chování jedince v prostředí je složen z nerůznějších akcí, odpovídajících způsobům jednání tohoto jedince v určitým způsobem charakteristických oblastech. Při znalosti pravidel a mechanismů syntézy výsledného chování, by ho tedy teoreticky

bylo možné dekomponovat. Nebo naopak, určitým způsobem optimální výsledné chování jedince (v tomto případě mobota) lze docílit vhodnou kombinací požadovaných jednání v jednotlivých specifických oblastech. Proto jsem i strukturu navrženého řídícího systému rozdělil do několika vrstev.

Celý řídící systém jsem navrhl pro předpokládanou implementaci jednotlivých vrstev pomocí prostředků, které ke své případné adaptaci budou využívat informaci pouze určitým způsobem ohodnocující jejich činnost, jako je tomu například u posilovaného učení (*Reinforcement learning*) neuronových sítí. Tento předpoklad nemá dle mého názoru omezující účinky, jelikož již získání tohoto druhu informace z prostředí je velmi obtížné a tudíž získání jakékoli specifičtější informace použitelné k adaptaci (jako např. správná odezva u učení s učitelem) téměř nemožné. Na konkrétní realizaci jednotlivých vrstev žádné další požadavky kladeny nejsou a tedy může být jejich implementace dokonce odlišná. Nyní již ale k samotné struktuře řídícího systému o něco podrobněji.

6.1.1 Globální struktura

Navržená globální struktura řídícího systému, bez vyznačení potřebných zpětných vazeb přes prostředí, je uvedena na obr. 21. Základem je rozdělení řízení do vrstev, z nichž každá zabezpečuje řešení dané třídy situací, a co je nejpodstatnější, každá má svoji úroveň priority. Každá z těchto vrstev tedy na základě příslušných senzorických dat navrhne, ze svého hlediska optimální, požadovanou akci (A_1, A_2, …, A_n). Modul správy akcí přijme tyto návrhy od všech vrstev a v závislosti na požadovaných akcích, na prioritách jednotlivých vrstev a případně na scénáři prováděného plánu (zde je nutno podotknout, že myšlenku plánování úloh jsem převzal z [Barnes, 1997]) vygeneruje akci výslednou. V dalším kroku pak modul zpracování hodnocení poskytne jednotlivým vrstvám příslušné hodnocení (*reinforcement* singál), které vygeneruje z vnitřního a popřípadě i externího hodnocení na základě informací o výsledné akci (IVA). Tyto hodnocení (z_1, z_2, …, z_n) využijí vrstvy řídícího systému k adaptaci.



obr. 21 - Globální struktura řídícího systému

Rozdělení vrstev, do uvedených základních oblastí z pohledu mobilní robotiky, samozřejmě není konečné. Každá z těchto vrstev může, a zpravidla i bude, obsahovat více než jeden modul, z nichž každý bude zase řešit pouze určitou podtřídu navzájem se prolínajících úloh z této specifické oblasti. Výstupem každé z vrstev pak tedy nebude jediná, ale určitý počet požadovaných akcí, které modul správy akcí bude muset odpovídajícím způsobem zpracovat.

6.1.1.1 Modul správy akcí

Úkolem modulu správy akcí je z množiny jednotlivými vrstvami požadovaných akcí nějakým způsobem vybrat akci výslednou.

Jednou z možností je to, že za výslednou akci prohlásí akci s největším stupněm priority. Takovýmto způsobem je možné vybrat výslednou akci mezi akcemi z různých vrstev, nikoliv však mezi akcemi pocházejícími z modulů (podvrstev) jedné vrstvy, v jejichž prioritách rozdíl není. Přiřazovat unikátní prioritu každému jednotlivému modulu, tj. v podstatě vytvořit z každého modulu samostatnou vrstvu, by tento problém sice vyřešilo, ale současně by to znamenalo omezení možností řídícího systému. V každém okamžiku, by totiž akce robota závisela na jednom konkrétním modelu jeho chování, interpretovaného danou vrstvou. Dále by muselo být zaručeno, že vrstvy z vyšší prioritou nebudou neustále generovat požadavky na akce, a že tedy i vrstvy z prioritou nižší budou mít někdy možnost se prosadit. Navrhnout jednotlivé vrstvy takovým způsobem, aby byly oblasti jejich působnosti zcela odděleny je pak v přímém rozporu z požadovanou komplexností výsledných reakcí.

Druhou, mnohem přijatelnější možností, je výslednou akci určitým způsobem z jednotlivých požadovaných akcí odvodit. V takovém případě už nebude třeba požadavky pocházející z různých vrstev a požadavky z více modulů jediné vrstvy nijak rozlišovat. Nejprve se množina všech navržených akcí rozdělí na skupiny souvisejících či odpovídajících si akcí, tzn. např. na množiny požadavků translace, rotace, komunikace, provedení měření speciálními senzory jako např. kamerou, apod. V rámci těchto jednotlivých skupin pak již bude možno výslednou akci jednoduše určit např. pouhým sečtením, vypočtením průměru, superpozicí či nějakou jinou metodou, přičemž při těchto operacích bude patřičným způsobem brán zřetel na prioritu každé z požadovaných akcí.

Vrstva zajišťující funkce přežití, tj. reflexní vazby, “vegetativní“ funkce apod., bude z důvodů výjimečnosti svého postavení od ostatních odlišena buďto výrazným odlišením velkosti priority, a nebo přidělením speciální úrovně priority. V prvém případě se tedy vliv ostatních požadavků téměř neprojeví, v druhém pak modul správy akcí při výskytu požadavku se speciální úrovní priority všechny ostatní akce zakáže.

Mechanismus plánovaní akcí, jehož myšlenku jsem převzal z [Barnes, 1997], spočívá v doplnění jednotlivých požadavků akcí o jejich kontext, tzn. o předcházející a následné podmínky. Tzn. že kompletní informace pro každou akci bude mít tvar [předcházející podmínky, akce, následné podmínky]. Modul správy akcí bude tedy navíc muset mít k dispozici informace, podle nichž požadované akce nejprve do tohoto tvaru doplní. Při samotném výběru budou pak navíc požadavky, jejichž předcházející podmínky kolidují s aktuálním stavem podmínek pro provedení akce, z výběru vyřazeny. Po provedení výsledné akce budou dle jejích následných podmínek nastaveny podmínky pro provedení akce další. Tímto způsobem pak mohou být některé akce v určitých situacích záměrně zcela potlačeny a realizovány tak, i jinak reaktivním způsobem řízení neproveditelné, sekvence reakcí.

Posledním, ale ne méně podstatným úkolem modulu správy akcí, je pak poskytnout modulu zpracování hodnocení, odpovídající informace o výsledné akci. Touto informací bude velikost podílu akcí jednotlivých vrstev na akci výsledné. Akce, které byly z určitého důvodu, a to ať už požadavkem se speciální úrovní priority, a nebo kontextem plánu, z výběru vyřazeny, budou tedy mít tento podíl nulový.

6.1.1.2 Modul zpracování hodnocení

Úkolem modulu zpracování hodnocení je zprostředkovat jednotlivým vrstvám odpovídající hodnotící (*reinforcement*) signály, které tyto vrstvy využijí k optimalizaci úspěšnosti svých návrhů, tj. ke své adaptaci. Výpočtem vnitřního hodnocení stavu systému, ze stavu baterií, energetického úbytku při provedení poslední akce, stavu specifických čidel (např. mechanických nárazníků) apod., a jeho kombinací s externím hodnocením, určí tento modul hodnocení celkové. To pak dle informací o podílu návrhů jednotlivých vrstev na výsledné akci, poskytnutých modulem správy akcí, příslušným způsobem mezi tyto vrstvy rozdělí. Vrstvy, jejichž návrhy byly z nějakého důvodu zamítnuty, a tedy následky provedené akce nijak s jejich činností nesouvisí, nebudou tudíž neadekvátně upravovat své rozhodovací mechanismy.

6.1.1.3 Moduly řídících vrstev

Moduly řídících vrstev generují na základě vstupních dat (aktuálního stavu prostředí) požadavky na vykonání akcí, které jsou z jejich pohledu určitým způsobem v dané situaci výhodné. Tyto moduly musí být, jak jsem již uvedl, implementovány pomocí prostředků, které ke své adaptaci využívají informaci pouze určitým způsobem ohodnocující jejich činnost, a to proto, že žádnou jinou informaci v uvedené struktuře řídícího systému k dispozici mít nebudou. Samozřejmě lze použít prostředků, které buďto své mechanismy nijak neoptimalizují a jichž lze použít např. při realizaci reflexních vazeb nejnižší úrovně, a nebo které ke své adaptaci využívají pouze vstupních dat.

Jednoduše řečeno, k realizaci jednotlivých modulů řídících vrstev lze použít jakýchkoli prostředků, k jejichž funkčnosti budou informace poskytované jim v této architektuře dostačující.

6.1.2 ASN

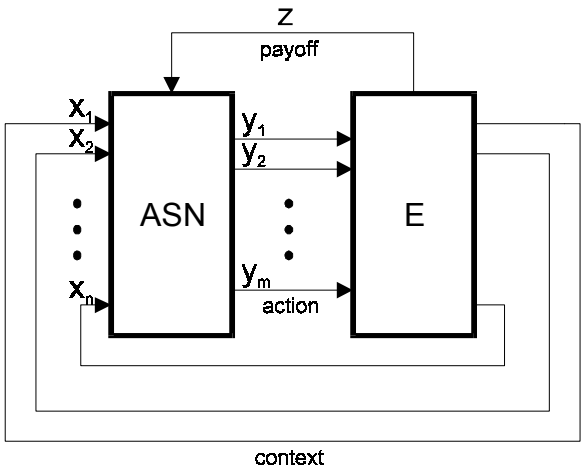
K vlastní realizaci vrstvy, resp. jejího modulu, umožňující koordinaci na nejnižší úrovni, což bylo hlavním úkolem této práce, jsem jako prostředku použil asociativně vyhledávací síť – Associative Search Network (ASN). V tomto odstavci, tedy popíšu její základní vlastnosti, strukturu a algoritmus učení.

Asociativně vyhledávácí síť, *Associative Search Network* (ASN), patří mezi neuronové sítě s posilovaným učením (*Reinforcement Learning*). Tato NS nepotřebuje pro učení množinu požadovaných asociací, tedy trénovací množinu, ale využívá při něm pouze tzv. hodnotící signál, který určitým způsobem ohodnocuje úspěšnost poslední provedené akce. Jedná se tedy o typ učení s učitelem známý jako učení s kritikou, nebo též učení na principu odměny a trestu.

ASN hledá pro každý vstupní vzor (vektor vstupů) takový výstupní vzor (akci), který hodnotící signál optimalizuje (maximalizuje). Tak jak učení probíhá, každý předložený vstupní vzor vyvolává nalezení lepší volby výstupního vzoru, tedy akce, která má být při daném stavu vstupů aktivována. Vlastní učení tedy spočívá v tom, že se do asociativní paměti ukládají výsledky zpětné vazby z prostředí. A to je taky největší výhodou ASN, jelikož pro svoji adaptaci nepožaduje předkládání explicitních vzorů trénovací množiny.

Žádná část systému tak nemusí mít žádné apriorní informace o "správných" přiřazeních akcí jednotlivým vstupním vzorům. Tato neuronová síť tedy současně řeší problém rozpoznávání vstupních vzorů a vyhledávání optimálního přiřazení akcí k nim.

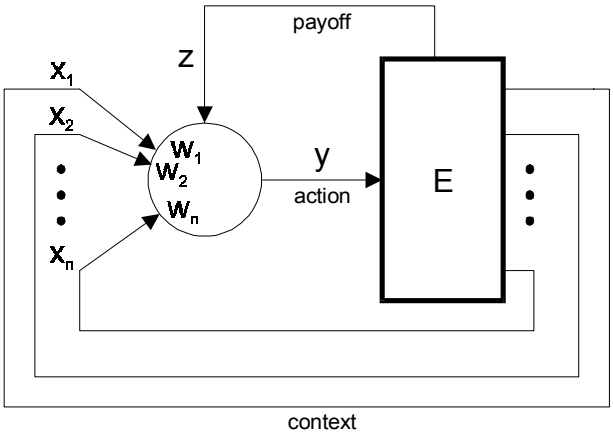
Princip interakce ASN z prostředím je patrný z obr. 22. V každém okamžiku má ASN k dispozici informaci o stavu prostředí ($x_1(t), x_2(t), \ldots, x_n(t)$) a hodnotící signál $z(t)$ (tzv. "payoff"). ASN na základě stavu prostředí (environment), resp. stavu svých senzorů, vygeneruje akci ($y_1(t), y_2(t), \ldots, y_m(t)$) a po jejím provedení, tzn. o jeden krok později, dostane ohodnocení provedení této akce v dané situaci (kontextu), a to prostřednictvím *payoff* signálu. Na základě této informace pak upraví váhy neuronů tak, aby aktivované akce v každém kontextu hodnotící signál *payoff* maximalizovaly.



obr. 22 - Interakce ASN a prostředí (E)

6.1.2.1 Neuron ASN

Asociativně vyhledávácí neuronová síť se skládá z množství neuronů, z nichž každý určuje jeden výstup výstupního vzoru. Každý neuron se tedy přímo podílí na zvolené akci. Vybavování a učení ASN tedy popíšu z důvodu názornosti na jediném neuronu, který lze považovat za nejjednodušší typ ASN. Interakce tohoto neuronu z prostředím je uvedena na obr. 23. Neuron má n vstupů ($x_1, x_2, \ldots, x_n$) a k nim asociovaných vah ($w_1, w_2, \ldots, w_n$) a jeden výstup y. Dále má k dispozici *payoff* signál z na jehož základě provádí korekci svých synaptických vah.



obr. 23 – Interakce neuronu ASN s prostředím (E)

Výstup neuronu asociativně vyhledávácí sítě v daném časovém okamžiku $y(t)$ je určen podle vztahu:

$$y(t) = \begin{cases} 1 & if \quad s(t) + NOISE(t) > 0 \\ 0 & jindy \end{cases},$$

$$s(t) = \sum_{i=1}^{n} w_i(t) x_i(t),$$

kde

$x_i(t)$ … je i-tý vstup neuronu v okamžiku t,

$w_i(t)$ … je váha i-tého vstupu neuronu v okamžiku t,

n … je počet vstupů neuronu,

$NOISE(t)$ … náhodná veličina s normálním rozdělením hustoty pravděpodobnosti a nulovou střední hodnotou.

Učení ASN, tedy korekce synaptických vah neuronu, je pak v každém kroku realizována úpravou vah dle vztahu:

$$w_i(t+1) = w_i(t) + \eta [z(t) - z(t-1)] \cdot [y(t-1) - y(t-2)] x_i(t-1),$$

kde

$w_i(t+1)$ … je nová (upravená) hodnota váhy i-tého vstupu neuronu,

$w_i(t)$ … je aktuální hodnota váhy i-tého vstupu neuronu,

η … je konstanta rychlosti učení,

$z(t), z(t-1)$ … jsou aktuální a minulá hodnota *payoff* signálu,

$y(t-1), y(t-2)$ … jsou minulá a předminulá hodnota výstupu,

$x_i(t-1)$ … je minulá hodnota i-tého vstupu neuronu.

Pro popis principu korekce vah je třeba si shrnout, že jednotlivé vstupy (kontext) nabývají pouze kladných reálných hodnot, *payoff* signál jakékoli reálné hodnoty a výstup pouze hodnoty 0, nebo 1. Za těchto předpokladů uvedu následující příklad.

Budu předpokládat, že vstup x_i v čase $t-1$ nabýval kladné hodnoty, což signalizovalo výskyt určitého stavu v prostředí. Dále budu předpokládat, že výstup $y(t-1) = 1$, zatímco $y(t-2) = 0$. To znamená, že v čase $t-1$ byl výstup tohoto neuronu aktivován, a to buď vlivem vstupu x_i a nebo díky náhodné veličině. Pokud se potom během přechodu z kroku $t-1$ do t zvětšila hodnota signálu *payoff*, a to možná právě díky změně výstupu tohoto neuronu v čase $t-1$, bude hodnota váhy w_i zvýšena.

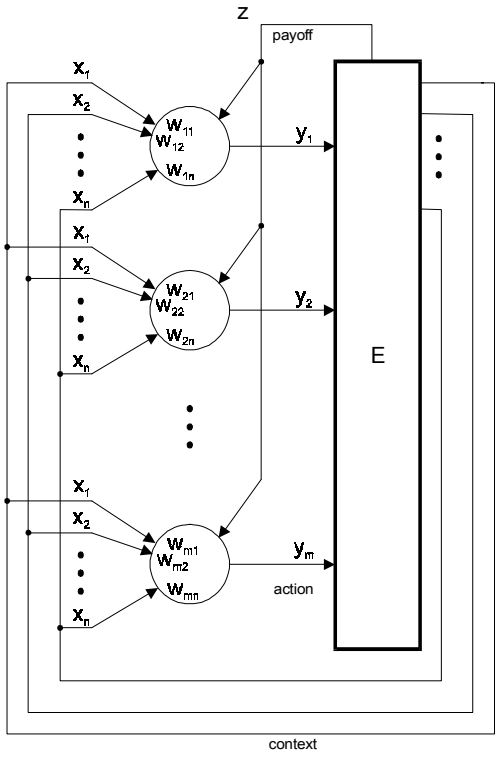
Jelikož se součin $w_i(t)x_i(t)$ podílí na hodnotě výstupu, bude zvýšení váhy w_i znamenat vetší pravděpodobnost, že bude výstup tohoto neuronu při výskytu daného stavu v prostředí opět aktivován. Naopak pokud bude změna výstupu neuronu následována snížením hodnoty signálu *payoff*, hodnota váhy w_i se sníží, což bude mít za následek snížení pravděpodobnosti aktivace tohoto neuronu při dalším výskytu signálu x_i, tj. v daném kontextu.

Jinými slovy, pokud je aktivace výstupu v určitém kontextu následována zvýšením *payoff* signálu, výstup neuronu bude v tomto kontextu aktivován (příp. ponechán na 1) v budoucnu s větší pravděpodobností. Stejně tak zvýšení *payoff* signálu při deaktivaci výstupu neuronu, bude mít za následek zvýšení pravděpodobnosti, že tento výstup bude v daném kontextu deaktivován i v budoucnu. Je zřejmé že ve svém důsledku spěje učení ASN k požadované maximalizaci *payoff* signálu.

V této chvíli je již zřejmý i význam složky *NOISE* podílející se na výpočtu hodnoty výstupu neuronu. Vliv této složky se bude projevovat dokud bude většina synaptických vah neuronu malá, což odpovídá situaci kdy neuron ještě neobsahuje téměř žádné informace. Neuron tak bude výstup (akci) generovat náhodně, což mu umožní si pomalu asociace vhodných akcí k příslušným kontextům začít vytvářet. Tím se ale pomalu některé z jeho vah zvýší a tudíž se při určité míře znalostí neuronu složka *NOISE* přestane uplatňovat.

6.1.2.2 ASN síť

Asociativně vyhledávací neuronová síť se skládá z množství neuronů popsaných v předchozím odstavci a její struktura je uvedena na obr. 24. Každému neuronu jsou zprostředkovány všechny složky kontextu a hodnotící signál *payoff*. Všechny neurony mají stejnou hodnotu konstanty učení. Náhodná veličina *NOISE* je pro každý neuron nezávislá, ale se stejným rozdělením a střední hodnotou. Podrobněji viz [Barto, 1981].



obr. 24 - Struktura ASN

6.1.3 Vrstva koordinace

Hlavním požadavkem na řídící systém bylo umožnění koordinace pohybu mobota s jiným mobotem, popřípadě skupinou mobotů. Potvrzením toho, že řídící systém, který jsem navrhl, tento požadavek splňuje, je realizace vrstvy umožňující koordinaci na nejnižší úrovni, jakou je například sledování. Nyní uvedu filosofii a strukturu této vrstvy a hlavní myšlenky, které jsem při jejím návrhu použil.

Jak jsem uvedl při popisu globální struktury řídícího systému, není jeho rozdělení na vrstvy dostačující a každá vrstva je tedy dále složena z modulů, z nichž každý řeší pouze určitou podtřídu úloh z dané, pro vrstvu specifické, oblasti. Realizace vrstvy umožňující koordinaci na nejnižší úrovni, tedy vlastně bude realizací jednoho z modulů vrstvy koordinace, zabezpečujícího řešení zmíněné třídy úloh. Proto v dalším textu budu používat označení modul, příp. modul koordinace.

Modul koordinace jsem realizoval pomocí asociativně vyhledávací neuronové sítě, jejíž princip jsem popsal v předcházejícím odstavci 6.1.2 - *ASN*. Nyní se tedy budu věnovat úpravám, které jsem při její implementaci provedl a k nimž mne inspirovaly především projekty [Kröse, 1998], [Barto(2), 1981] a [URL, University of Southern California]. Modul koordinace bude tedy ke své adaptaci využívat algoritmy posilovaného učení. Důležitým poznatkem zmíněných projektů je, že dodáním určité obecné znalosti určité oblasti, a to ať už ve formě tzv. "progress estimator" (průběžného odhadu) viz [Matarić, 1997], a nebo využitím orientačních majáků viz [Barto(2), 1981], výrazně zvýšíme rychlost učení a urychlíme konvergenci řešení. Projekt [Kröse, 1998] zase ukazuje výhody lokalizace v prostoru senzorických dat.

Jako demonstrační úloha koordinace mobotů se naprosto zřejmě nabízí koordinace jejich vzájemného pohybu. Podrobnosti k vlastnímu návrhu úlohy uvedu v následujícím odstavci. Je jasné, že pro úspěšné vykonávání koordinace vzájemného pohybu, budou moboti muset mít určité znalosti o svých vzájemných polohách. Kromě hodnotícího *payoff* signálu, který je externím zdrojem informace o vhodnosti vzájemné polohy, a který moboti získávají na základě komunikace, jsou, právě pro urychlení konvergence učení, rozšířeni o senzory, umožňující jim navíc získávat potřebné pomocné informace (obecné znalosti). Zde jsem se výrazně zaměřil na to, aby takto doplněné zdroje informace byly realizovatelné, a současně aby nebyly nijak svázány se samotným prostředím. To totiž umožní případné pozdější využití možností oblasti orientace a plánování akcí v prostoru senzorických dat.

Moboty jsem tedy rozšířil (v oblasti simulace) o modul vzájemné komunikace prostřednictvím tabule a o senzory viditelnosti ostatních mobotů. Modul komunikace slouží pro získání hodnotícího signálu a senzory viditelnosti ostatních mobotů určují, jak již jejich název napovídá, ve kterých směrech jsou v daném okamžiku viditelní ostatní moboti. Bližší informace o obou těchto modulech uvedu později v kapitole 6.3 - *Implementace a dosažené výsledky*.

Právě senzory viditelnosti ostatních mobotů byly důvodem rozšíření typického neuronu ASN o další vstup. Je zřejmé, že v určitém okamžiku nemusí být v okolí mobota, resp. v dosahu zmiňovaných senzorů, přítomen žádný robot. Pak by hodnoty všech těchto senzorů, použité jako jediné vstupy ASN, byly nulové. Jak víme z popisu ASN, převládne v takovém okamžiku náhodná složka *NOISE*, která se podílí na výstupu a síť začne generovat požadavky na náhodné akce. Mobot se tedy začne pohybovat náhodným způsobem, což je při této lokální nepřítomnosti ostatních robotů, a tedy situaci kdy není s kým koordinovat svůj pohyb, z hlediska vrstvy koordinace výhodné. Je to koneckonců cesta k nalezení ostatních robotů k možné koordinaci. Samozřejmě to tak nemusí být z pohledu jiných vrstev a jejich modulů řídícího systému, které mohou

v této situaci požadovat jiný výsledný pohyb. To ale může být vyřešeno již samotnou strukturou řízení, přesněji již probraným modulem správy akcí. Jelikož budu realizovat pouze jediný modul vyššího řízení, nebudu se konkrétním řešením této situace modulem správy akcí zabývat.

To na co se chci zde zaměřit, je skutečnost, že i při nulových hodnotách senzorů viditelnosti ostatních robotů, je možné dále určitým způsobem koordinovat pohyb mobota, a to čistě na základě *payoff* signálu, pokud je tento dostupný. To nám umožní právě rozšíření neuronů ASN o další (nultý) vstup, jehož hodnota bude trvale nastavena na 1 a váhy příslušející tomuto vstupu u jednotlivých neuronů upravovány algoritmem založeným na stejném principu jako algoritmus úprav vah standardních. Rozdílem bude jen jejich vlastní hodnota konstanty učení, omezení velikosti hodnot těchto vah a samozřejmě nepatrná úprava výpočtu výsledných výstupů neuronů. Váhy příslušející diskutovanému speciálnímu vstupu budou tedy upravovány dle vztahu:

$$w_{0j}(t+1) = f\left[w_{0j}(t) + \eta_0\,[z(t) - z(t-1)] \cdot [y_j(t-1) - y_j(t-2)]\right],$$

$$f(x) = \begin{cases} MEZ & if \quad x > MEZ \\ 0 & if \quad x < 0 \\ x & jindy \end{cases},$$

kde

$w_{0j}(t+1)$	… je nová (upravená) hodnota váhy 0-tého vstupu j-tého neuronu,
$w_{0j}(t)$	… je aktuální hodnota váhy 0-tého vstupu j-tého neuronu,
η_0	… je konstanta rychlosti učení,
$z(t), z(t-1)$	… jsou aktuální a minulá hodnota *payoff* signálu,
$y_j(t-1), y_j(t-2)$	… jsou minulá a předminulá hodnota výstupu j-tého neuronu,
MEZ	… je maximální hodnota váhy.

Maximální hodnota váhy je stanovena proto, aby při pohybu ve směru rostoucího signálu *payoff* tato příliš nevzrostla a bylo ji možno při pohybu ve směru klesajícího signálu *payoff* opětovně snížit na nulu. Také tím bude zaručeno, že při přítomnosti informace z pomocných senzorů viditelnosti ostatních robotů neměla tato složka na výstup vliv. Výstupy jednotlivých neuronů ASN v daném časovém okamžiku $y_j(t)$ pak tedy budou určeny podle vztahu doplněného o příspěvek nultého vstupu:

$$y_j(t) = \begin{cases} 1 & if \quad s_j(t) + NOISE_j(t) > 0 \\ 0 & jindy \end{cases},$$

$$s_j(t) = w_{0j}(t) + \sum_{i=1}^{n} w_{ij}(t) x_i(t),$$

kde významy jednotlivých členů jsou patrné z popisů předcházejících rovnic.

Jak jsem vysvětlil v odstavci 6.1.2 - *ASN*, člen $[y(t-1)-y(t-2)]$ v rovnici úpravy vah neuronů zajišťuje to, aby k odpovídajícím korekcím docházelo pouze při změnách příslušného výstupu, tzn. nastavení na 1 (aktivaci) a nastavení na 0 (deaktivaci). Jinými slovy jsou tak sledovány pouze vlivy změn výstupů neuronů na změny v hodnotě signálu *payoff*.

Abych přínos sledování změn výstupu mohl v simulaci demonstrovat, umožnil jsem při vlastní realizaci také úpravu vah dle vztahu, ve kterém je zmiňovaný člen nahrazen pouze hodnotou $y(t-1)$. Takto upravený vztah pro korekci vah tedy je:

$$w_{ij}(t+1) = w_{ij}(t) + \eta[z(t) - z(t-1)]\, y_j(t-1)\, x_i(t-1),$$

kde významy jednotlivých členů jsou patrné z popisů předcházejících rovnic.

Stejným způsobem je pak upraven i vztah pro korekci vah nultého vstupu. Při použití tohoto upraveného tvaru bude tedy k odpovídajícím korekcím docházet vždy, když bude hodnota příslušného výstupu rovna jedné. Jinými slovy nebudou tak u jednotlivých neuronů sledovány vlivy změn výstupů, ale přímé působení výstupů na změny v hodnotě *payoff*.

V programu simulace lze mezi uvedenými variantami algoritmů korekce vah jednoduchým způsobem volit, což mi umožní demonstrovat přínos prvého způsobu korekce vah, tedy sledování změn.

6.2 Demonstrační úloha

Jako demonstrační úloha koordinace mobilních robotů se naprosto zřejmě nabízí koordinace jejich vzájemného pohybu. V tomto odstavci uvedu tedy podrobněji jednotlivé cíle, které jsem si v této oblasti stanovil a potřebné předpoklady a omezení, které tyto kladou jak na vybavení samotného mobota, tak na prostředí, ve kterém se bude mobot pohybovat. Kromě cílů stanovených a dosažených v rámci této diplomové práce, uvedu dále možnosti, v nichž vidím další možné cesty vývoje, ke kterým jsem směřoval a kterým bych se chtěl dále věnovat v rámci doktorandského studia.

Demonstrační úlohu jsem soustředil, tak jak bylo úkolem této práce, na předvedení možností koordinace pomocí navrženého řídícího systému mobota, tedy především na ověření schopností modulu koordinace. Z tohoto důvodu jsem nepostupoval od základní implementace reflexních vazeb, přes prostředky zajišťující vyhýbání se překážkám až směrem k vlastnímu modulu koordinace, ale naopak jsem se na modul koordinace zaměřil rovnou. Proto jsou počáteční omezení prostředí velmi zásadní. Znamená to, že jsem předpokládal prostředí neomezené velikosti a bez přítomnosti jakýchkoli překážek, což mi umožnilo soustředit se na odladění vlastních algoritmů koordinace. Samozřejmě, že již od počátku jsem implementoval všechny prostředky umožňující pozdější realizaci dalších vrstev řízení, přičemž jsem se držel technických parametrů reálných robotů.

Jako hlavní cíl jsem si v rámci této práce stanovil úlohu stíhání či "lovu" jednoho mobota druhým. Jedná se vlastně o úlohu udržování určité specifické polohy vůči druhému robotovi, v tomto případě tedy co "nejbližší", jež je z hlediska koordinace, a nakonec i z vyššího hlediska případné kooperace, v oblasti mobilní robotiky jednou

z nejpodstatnějších. Tato úloha rovněž skýtá možnosti dalšího rozvoje, a to v rozšíření na udržování určitým způsobem specifické polohy vůči více robotům současně, a tedy realizaci např. stíhání či "lovu" jednoho mobota celou skupinou mobotů dalších. Nastínění realizace takovéto úlohy je v odstavci 6.4 - *Možné další směry vývoje*.

Postup práce na samotné realizaci demonstrační úlohy rozdělím do tří etap, z nichž první, *Prostředí simulátoru a úloha dosažení polohy statického cíle*, je etapou nejpodstatnější a nejrozsáhlejší. Druhá etapa, *Stíhání dynamického cíle*, je dovršením stanoveného cíle této práce a etapa poslední, *Rozšíření o základní reflexní vazby*, realizovaná již nad rámec vlastního zadání, je rozšířením celé úlohy o možnosti realizace v prostředí z překážkami. V následujících odstavcích stručně shrnu náplň zmíněných etap a stejným způsobem, tedy rozdělením podle těchto etap, budu posléze koncipovat i odstavec 6.3 - *Implementace a dosažené výsledky*.

6.2.1 Prostředí simulátoru a úloha dosažení polohy statického cíle

V této, první etapě, jsem se věnoval vytvoření vlastního prostředí simulátoru v programovacím jazyce C++ a objektu (ve smyslu základního prostředku objektového programování) mobota, u kterého jsem implementoval senzorické zdroje informací odpovídající vybavení reálných mobotů, tj. mechanické nárazníky, infračervená čidla (IR čidla) a sonar. Zaměřil jsem se a implementoval všechny potřebné metody snadné modifikace všech základních parametrů tohoto objektu mobota, jakými jsou velikost mobota, počet mechanických nárazníků, počet, dosah a rozlišení IR čidel apod. Dále jsem tento objekt rozšířil o dříve diskutované senzory viditelnosti ostatních mobotů.

Poté jsem vytvořil objekt asociativně vyhledávací neuronové sítě (ASN), jež je opět z hlediska počtu vstupů, výstupů a dalších parametrů, snadno modifikovatelný a jež byl základem pro následné vytvoření a rozšíření řízení mobota o modul koordinace.

Dokončením této etapy, pak byla realizace a odladění vlastní úlohy stíhání statického cíle, tedy vlastně dosažení jeho polohy, a to především odladění parametrů neuronové sítě a senzorů viditelnosti ostatních mobotů.

6.2.2 Stíhání dynamického cíle

V této etapě, jsem rozšířil objekt mobota o potřebný modul umožňující komunikaci prostřednictvím tabule a samotný simulátor tedy nutně o základní prostředek tohoto způsobu komunikace, o vlastní tabuli. Komunikace mezi roboty, je nutná jen z důvodu realizace senzorů viditelnosti druhého robota, kdy je viditelnost robota jednotlivými senzory počítána ze vzájemné polohy a okamžitých úhlů natočení obou robotů. Při implementaci reálnými roboty by tato informace byla získána z hodnot příslušných senzorů a tudíž by teoreticky nebyla komunikace mezi moboty nutná.

Nicméně ještě jedna informace je na komunikaci robotů závislá, a to sice externí hodnotící signál *payoff*, který moboti počítají také z velké části na základě své vzájemné polohy. Pokud by v reálné aplikaci tuto informaci, *payoff* signál, poskytoval nějaký externí zdroj, komunikace mezi moboty by pak skutečně nutná nebyla. Realizace takového externího pozorovatele by ale v reálném prostředí byla téměř nemožná. Proto bude dle mého názoru jednodušší, když si moboti navzájem vymění informace o svých polohách, a signál *payoff* si vypočtou dle předem stanovených pravidel, pomocí těchto a případně i dalších informací, získaných například přímo prostřednictvím senzorů.

Závěrem této etapy, pak bylo odladění a odzkoušení úlohy zadání, tedy provedení simulací stíhání jednoho robota druhým a otestování vlivu změn poloh cíle na průběh učení použité neuronové sítě ASN.

6.2.3 Rozšíření o základní reflexní vazby

Posledním rozšířením, které jsem považoval za podstatné provést, bylo rozšíření řídícího algoritmu mobota alespoň o nejnižší úroveň reflexních vazeb. Tuto jsem realizoval nejjednodušším způsobem, a to tak, že při detekování překážky mobot couvne, otočí se o daný úhel a v tomto směru udělá další krok. Pokud poté už k nárazu nedojde, převezme řízení opět modul koordinace. Toto rozšíření jsem provedl jen pro demonstraci funkčnosti řídícího systému při implementaci další vrstvy, a z tohoto důvodu jsem také žádný "Inteligentnější" způsob objíždění překážek nevyvíjel.

6.3 Implementace a dosažené výsledky

V této kapitole popíšu vlastní implementaci a uvedu výsledky simulací, které jsem v jednotlivých etapách postupu řešení provedl. Tato kapitola je tedy přehledem dosažených výsledků a podle toho je též koncipována. K vlastní implementaci programu simulátoru uvedu jen základní charakteristiky, jelikož programovací techniky, přesto že tvořily nejpodstatnější část vlastní práce, byly vlastně jen nutným prostředkem a nejsou náplní tématu tohoto projektu.

6.3.1 Simulátor

Program jsem vyvinul pro prostředí MS Windows 9x/NT a napsal v jazyce C++ ve vývojovém prostředí Borland C++ Builder 4.0. Program jsem se pokusil koncipovat tak, aby byl přehledný a lehce ovladatelný, bez jakékoliv nápovědy.

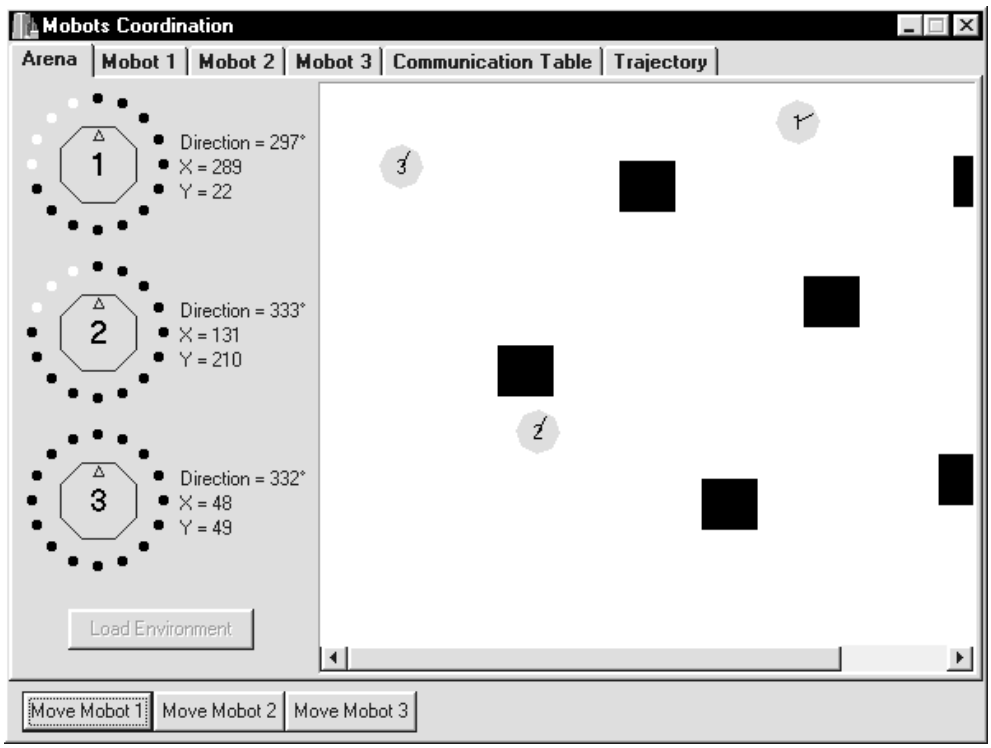
Celý program se při běhu skládá z několika paralelně běžících procesů (vláken). Tím hlavním je vlastní prostředí simulátoru, zprostředkovávající komunikaci s uživatelem, a to ve smyslu vizualizace a přístupu k parametrům jednotlivých objektů a ovládání běhu prováděných simulací. Také obsahuje prostředky interakce mobotů s prostředím a prostředky pro realizaci jejich vzájemné komunikace. Zbylými procesy jsou vlákna objektů jednotlivých mobotů, jež mají implementovány všechny potřebné metody pro interakci z prostředím, pro komunikaci prostřednictvím tabule a především všechny metody vlastního řídícího systému mobota a zdrojů senzorických informací. Součástí objektů jednotlivých mobotů je samozřejmě objekt asociativně vyhledávací sítě, jež tvoří součást jejich řídícího systému. Všechny paralelně běžící procesy jsou odpovídajícím způsobem synchronizovány, tak aby nedocházelo ke konfliktům při přístupu k potřebným zdrojům dat a vlastní prostředí simulátoru je zabezpečeno stejným způsobem. To znamená že neumožňuje uživateli například za běhu vlákna mobota měnit jeho technické parametry apod.

Podrobnější výklad týkající se použitých programovacích technik zde, jak jsem již zdůvodnil, uvádět nebudu. V dalším textu se tedy zaměřím na popis funkce příslušných struktur a nikoliv na výklad jejich vlastní implementace. Přesto jsem pouze pro bližší představu, a tedy bez dalších podrobností, uvedl vztah použitých objektů a výpis jejich hlavičkových souborů v přílohách 1-8. Nyní tedy nejprve popíšu vlastní prostředí simulátoru a základní informace k jeho ovládání, poté popis objektu asociativně vyhledávací neuronové sítě ASN, která je součástí objektu mobota a zajišťuje řízení jeho pohybu a nakonec objekt mobota a jeho, z hlediska této práce, podstatné metody. Při tomto popisu budu především z důvodu názornosti používat obrázky reálné aplikace, z nichž budou jednotlivé vlastnosti a současně i způsoby nastavení příslušných parametrů, stejně jako samotný princip ovládání celého programu, patrné nejlépe.

6.3.1.1 Vlastní prostředí simulátoru

Samotné prostředí simulátoru zprostředkovává potřebnou komunikaci s uživatelem a umožňuje mu přístup k nastavení parametrů jednotlivých objektů a podmínek, za kterých bude provedena příslušná simulace. Důležitou součástí jsou implementované prostředky interakce mobotů s prostředím a prostředky pro realizaci vzájemné komunikace mezi moboty. V tomto prostředí jsou implementováni tři moboti, jelikož tento počet byl pro zamýšlené simulační úlohy dostačující. Nicméně rozšíření o další moboty je díky objektovému přístupu realizovatelné velmi jednoduše.

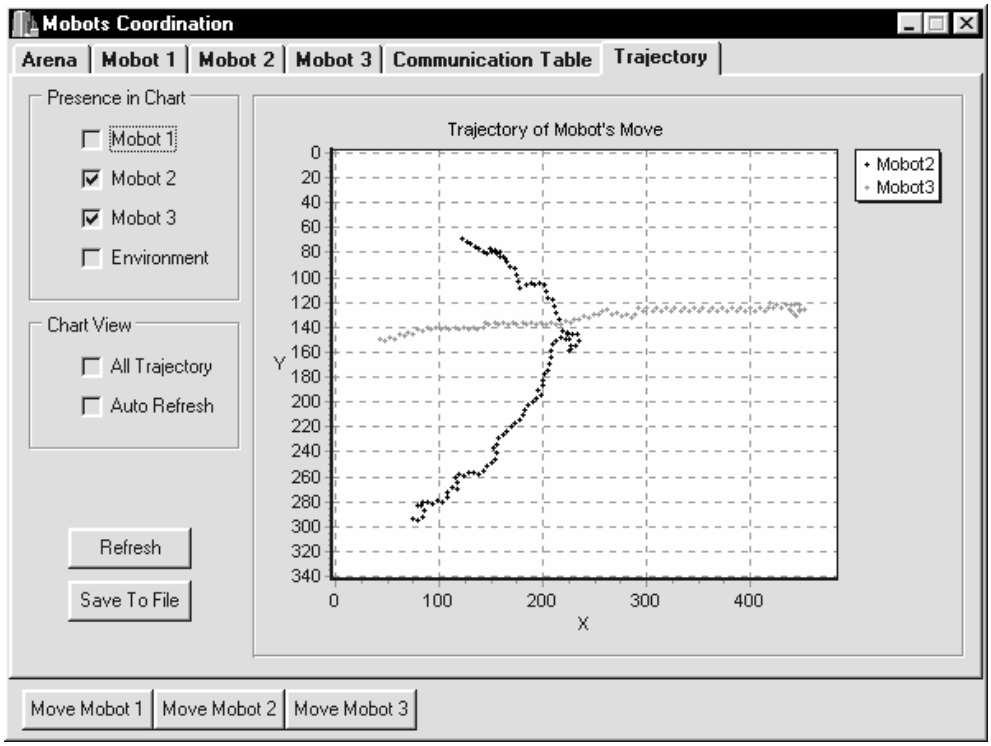
Vizuální prostředí simulátoru jsem rozdělil do několika hlavních částí. První částí je panel vizualizace stavů senzorů mobotů a vlastního prostředí (*Arena*), viz obr. 25. Je zde možno načíst mapu prostředí, jež je reprezentována obrázkem ve formě bitmapy. Dále zde jsou zobrazovány pohyby mobotů v prostředí, aktuální souřadnice jejich pozice, úhel natočení a okamžité stavy jejich senzorů, tj. mechanických nárazníků (úsečky obrysu robota) a IR čidel (vyplněné kružnice rozmístěné rovnoměrně kolem mobota). Z obr. 25 je též vidět, že při přítomnosti mobotů v prostředí nelze měnit prostředí samotné (tlačítko *Load Environment* není aktivní). Třemi tlačítky zcela dole (*Move Mobot 1 - 3*), lze jednoduše přerušit a znovu spustit pohyby jednotlivých mobotů. Jak dále uvidíme, jsou tyto tlačítka přístupná ze všech panelů simulátoru.



obr. 25 - Panel vizualizace pohybu mobotů a stavů jejich senzorů

Dalším panelem zaměřeným zcela na vizualizaci stavu prostředí a mobotů je panel zobrazení trajektorií pohybu mobotů (*Trajectory*), viz. obr. 26, pomocí něhož byly vytvořeny všechny obrázky trajektorií mobotů, které budu uvádět později při popisu provedených silmulací. Na panelu lze jednoduchým způsobem volit zobrazení trajektorií pohybu jednotlivých mobotů, přítomnost obrazu prostředí na pozadí, zobrazit

celou trajektorii mobota, tzn. příslušným způsobem zvětšit či zmenšit měřítko grafu a tím i zobrazenou oblast (zoom) a také přímo pozorovat aktuální pohyby mobotů po zapnutí automatické obnovy grafu (*Auto Refresh*). Příslušný graf pak lze uložit do souboru ve formátu bitmapy. Ve spodní části jsou opět zpřístupněna tlačítka, jimiž lze jednoduše přerušit a znovu spustit pohyby jednotlivých mobotů.

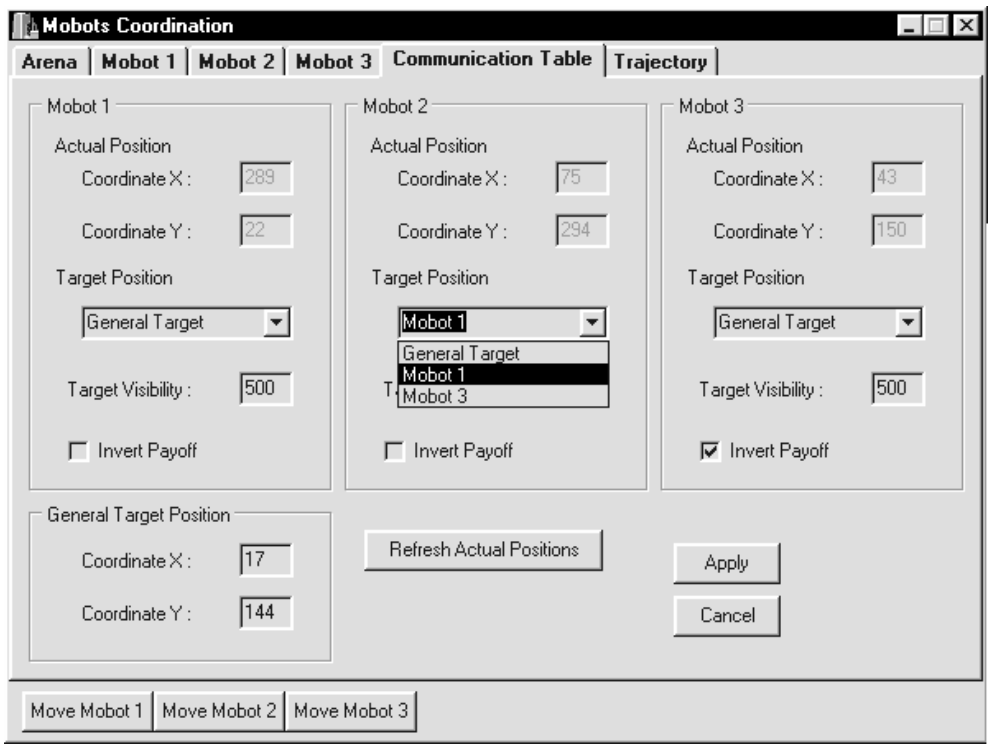


obr. 26 - Panel zobrazení trajektorií pohybu mobotů

Jednou z částí již přímo ovlivňujících parametry řízení jednotlivých robotů je panel **komunikační tabule** (*Communication Table*), viz obr. 27. Při komunikaci realizované prostřednictvím tabule, jednotliví agenti zapisují do síleného prostředku (tabule) informace, které chtějí zveřejnit. Naopak si z této tabule mohou přečíst informace, o které mají zájem. Přístupy k jednotlivým položkám komunikační tabule však musí být synchronizovány, aby nemohlo dojít k dezinformaci. To by se mohlo stát v případě, když by jeden agent četl z tabule informace, které jiný agent v tutéž chvíli zrovna přepisuje. Výsledkem by pak mohla být naprosto zkreslená informace. Tomu je zabráněno právě synchronizací přístupu agentů k jednotlivým položkám tabule, a to tak, že pokud do dané položky tabule určitý agent něco zapisuje, je všem ostatním agentům přístup k této položce zakázán. Agenti požadující čtení této položky tedy musí počkat, dokud jim opět, nebude povolen přístup. To je učiněno poté, co zmíněný agent ukončí zápis informací do této položky. Naopak pokud nějaký agent z určité položky komunikační tabule čte, je zamítnut přístup všem ostatním agentům, kteří chtějí do této položky zapisovat. Agentům požadujícím její čtení je samozřejmě přístup povolen, jelikož zde konflikt nehrozí. Agentovi požadujícímu zápis do dané položky je přístup zamítnut do doby, kdy všichni agenti provádějící její čtení, tuto činnost ukončí. A přesně takovýto způsob synchronizace jsem u komunikace mezi jednotlivými moboty implementoval.

Informacemi, které moboti zapisují na tabuli, jsou aktuální souřadnice jejich okamžité polohy. Naopak informacemi, které moboti z tabule čtou, jsou aktuální souřadnice okamžité polohy svého cíle, tzn. objektu vzhledem k němuž mají za úkol koordinovat svůj pohyb. Tímto objektem může být buďto jiný mobot, což je cílem této práce, a nebo statický cíl (*General Target*), jehož jsem využíval v první etapě práce pro realizaci úlohy dosažení statického cíle. Cílový objekt mobota lze nastavit pomocí rozbalovacího menu (*Target Position*), které nabízí možné cílové objekty viz. obr. 27. Dále jde prostřednictvím tohoto panelu nastavit viditelnost cíle (*Target Visibility*), což v podstatě znamená dosah senzorů viditelnosti ostatních mobotů, a zapnout invertování funkce *payoff* (*Invert Payoff*), což umožní jednoduchým způsobem změnit cílové chování robota snažícího se k cíli přiblížit, naopak na snahu se od tohoto cíle co nejvíc vzdálit, což umožní realizovat útěk tohoto mobota před mobotem "lovcem". Posledními parametry, které lze prostřednictvím tohoto panelu měnit, je poloha statického cíle (*General Target Position*).

Neaktivní políčka v horní části, označená jako aktuální pozice mobotů (*Actual Position*), zobrazují obsah proměnných zabezpečených synchronizací přístupu, jejichž prostřednictvím moboti komunikují. To znamená proměnných, do nichž moboti zapisují informace o své poloze a z nichž čtou informace o poloze jim určeného cíle. Tyto políčka nejsou obnovována průběžně, ale jejich obnovení lze v případě potřeby provést manuálně (stiskem tlačítka *Refresh Actual Positions*). Uživatel samozřejmě tyto položky měnit nemůže. Jinak je tomu u pozice statického cíle, u něhož je zadání jeho polohy je na uživateli přímo požadováno. To lze učinit zápisem do příslušných políček a stiskem tlačítka *Apply*, a nebo též kliknutím na příslušnou pozici v prostředí, a to sice v panelu vizualizace stavů senzorů mobotů a vlastního prostředí. Tlačítko *Apply* současně potvrdí všechny změny provedené v tomto panelu.



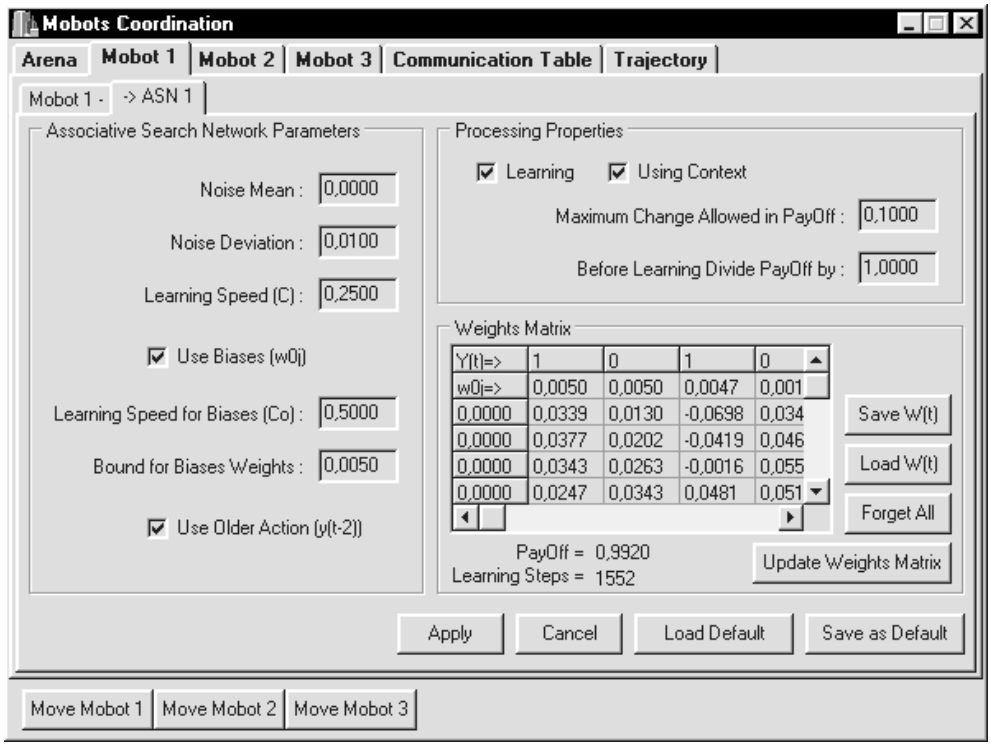
obr. 27 - Panel komunikační tabule

6.3.1.2 Objekt ASN

Důležitou součástí řídícího systému každého z mobotů je asociativně vyhledávácí síť (ASN). Vytvořil jsem tedy třídu TAssociativeSearchNetwork, z níž jsou poté samotné objekty ASN sítě mobotů odvozeny. Výpis hlavičkového souboru této třídy je uveden v příloze 3.

Objekt asociativně vyhledávácí sítě jsem záměrně vytvořil zcela modifikovatelný. Tento objekt zprostředkovává všechny funkce ASN tak, jak jsem je uvedl při popisu řídícího systému v odstavcích 6.1.2 a 6.1.3. Objekt ASN umožňuje libovolně měnit strukturu sítě, resp. počet jejích vstupů a výstupů, a jednoduchým způsobem zprostředkovává metody vybavování a učení. Z vnějšího pohledu tedy stačí pouze zadat počet vstupů a výstupů, potřebné parametry učících algoritmů a dále již jen zadávat hodnoty vstupů a signálu *payoff* a patřičným způsobem využívat výstupů sítí nám poskytnutých. O vlastní správu ASN, tj. o přípravu pro další krok, o provedení korekce synaptických vah neuronů apod., se objekt již postará "sám", prostřednictvím vnitřních metod, jež jsem v něm implementoval.

Součástí prostředí simulátoru je tedy samozřejmě i panel správy ASN pro každého mobota (*Mobot 1-3 – ASN 1*), viz obr. 28. Jeho prostřednictvím je lze jednak nastavovat parametry algoritmů učení a vybavování sítě, jako jsou konstanty rychlosti učení (*Learning Speed (C)* a *Learning Speed for Biases (C_0)*) a parametry náhodné veličiny *NOISE* (*Noise Mean* a *Noise Deviation*), jednak přímo ovlivňovat činnost ASN např. určením, zda má být použit nultý vstup ASN (*Use Biases (w0j)*), stanovením, zda má být při učení použit člen *y(t-2)*, tedy sledovány změny výstupů, (*Use Older Action (y(t-2))*), zda má být prováděna korekce vah (*Learning*), zda má být brán zřetel na vstupy kontextu, tedy na hodnoty senzorů viditelnosti ostatních mobotů, (*Using Context*) apod. Současně jsou v tomto panelu z důvodu kontroly činnosti zobrazovány hodnoty jednotlivých vah, vstupů a výstupů ASN (*Weight Matrix*), aktuální hodnota signálu *payoff*, počet provedených kroků učení (*Learning Steps*). Lze také ukládat a posléze znovu načíst všechny nastavení parametrů ASN a rovněž tak hodnoty vah sítě.

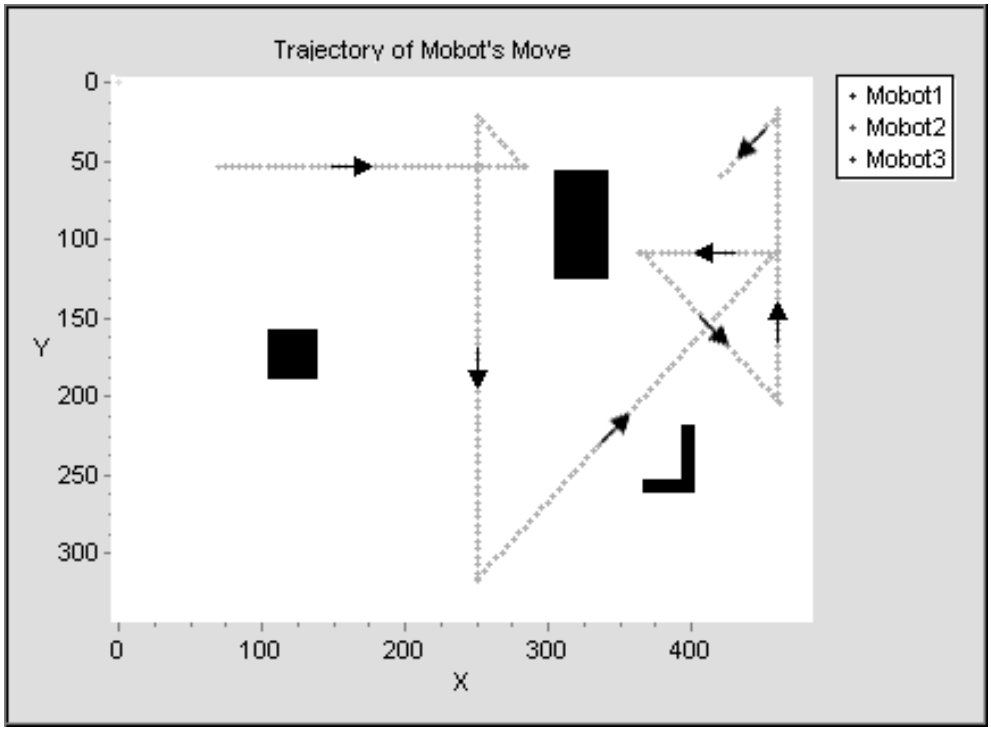


obr. 28 - Panel správy ASN

6.3.1.3 Objekt Mobota

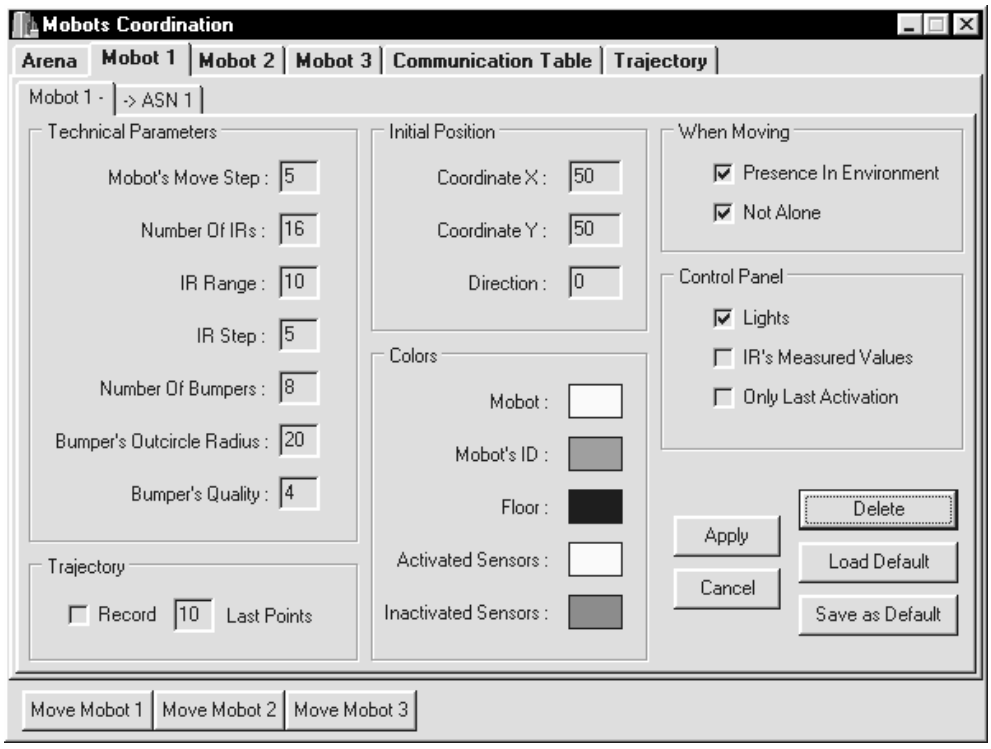
Hlavím prvkem celého simulátoru je samozřejmě vlastní interpretace mobota. Pro tento účel jsem vytvořil třídu TMobotThread, která dědí metody nutné pro umožnění běhu v samostatném vlákně (paralelně s metodami simulátoru a ostatních mobotů) od třídy TThread, která je standardní součástí knihovny VCL (Visual Component Library). Třídu TMobotThread jsem navrhl tak, aby v maximální míře umožňovala snadnou modifikaci objektu, z hlediska technických parametrů mobota a jeho senzorů. Ve třídě TMobotThread jsem implementoval všechny potřebné metody zabezpečující vlastní simulaci interakce mobota z prostředím, tj. získávání senzorických informací, komunikaci s ostatními moboty, modifikaci technických parametrů mobota a v neposlední řadě vizualizaci. Také jsem zde deklaroval metody řídícího systému mobota, ale tyto metody jsem již přímo neimplementoval. Tato třída je totiž určena pro základ tvorby dalších tříd, v nichž jsou všechny základní metody zděděny a implementovány právě již jen vlastní metody řízení. Tzn. že při další práci jsem vždy vytvořil novou třídu a implementoval příslušné prostředky řízení.

První takovouto třídou, určenou pro otestování funkčnosti metod senzorických měření, byla TSimpleMobotThread. Výpis jejího hlavičkového souboru je uveden v příloze 4. V této třídě jsem implementoval nejjednodušší způsob řízení pohybu mobota, a to takový, že při nárazu na překážku mobot jednoduše couvne, otočí se o přesně stanovený úhel (+135° tzn. doleva) a v tomto novém směru pak pokračuje dále ve svém pohybu. Trajektorie pohybu mobota je patrná z obr. 29. Po odladění funkčnosti metod senzorických měření jsem vytvořil třídu TMobotASNThread, do níž jsem již zahrnul i objekt asociativně vyhledávací sítě (ASN). Všechny zbývající použité třídy jsem poté odvodil od třídy TMobotASNThread a již jen implementoval příslušným způsobem upravené metody řízení. Vzájemný vztah všech použitých tříd objektů je uveden v příloze 1 a výpisy jejich hlavičkových souborů v přílohách 2-8.



obr. 29 - Trajektorie pohybu mobota třídy TSimpleMobotThread

Poslední součástí vizuálního prostředí simulátoru je panel parametrů mobota (*Mobot 1-3 – Mobot 1-3*), který je uveden na obr. 30. Prostřednictvím tohoto panelu lze nastavovat technické parametry mobota, jimiž jsou krok mobota (*Mobot's Move Step*), počet IR čidel (*Number Of IRs*), jejich dosah (*IR Range*) a rozlišení (*IR Step*), počet mechanických nárazníků (*Number Of Bumpers*) a jejich citlivost (*Bumper's Quality*) a poloměr mobota (*Bumper's Outcircle Radius*). Dále zde lze nastavit zda (*Record*), a v případě že ano tak kolik, posledních bodů (*Last Points*) svého pohybu si má mobot pamatovat. Tyto jsou poté použity pro vizualizaci jeho trajektorie. Možnost nastavení počáteční polohy mobota (*Initial Position*), barvy, kterou robot považuje za volné místo v prostředí (*Floor*) a barev týkajících se jeho vizualizace, je samozřejmostí. Zbývající uvedené parametry, jimiž lze nastavit například způsob zobrazování hodnot IR čidel (sonaru) v panelu vizualizace pohybů mobotů a stavů jejich senzorů, jsem využíval především pro odlaďování implementovaných metod senzorických měření. Změny v nastavení se stejně jako ve všech ostatních panelech projeví až po jejich potvrzení tlačítkem (*Apply*). Zrušit objekt mobota a posléze vytvořit nový lze pomocí tlačítka (*Delete*). To je taky jediný způsob, jak mobota již existujícího v prostředí, z tohoto prostředí odebrat. Některá nastavení jsou přístupná pouze do doby, kdy je mobot umístěn do prostředí prvním požadavkem ne jeho pohyb. Měnit jeho parametry v době kdy se již vyskytuje v prostředí, a kdy není zaručeno, že v jeho blízkosti není např. další mobot a nebo překážka, by mohlo vést ke konfliktům. Nicméně měnit technické parametry mobota po jejich počátečním nastavení již není nutné a z tohoto důvodu není použitý způsob zamezení konfliktům omezující. Samozřejmě že je zde také umožněno všechny nastavení uložit nebo načíst.



obr. 30 - Panel parametrů mobota

Programovací techniky samotné implementace jednotlivých metod zde, jak jsem již zdůvodnil, uvádět nebudu. Nicméně pro úplnost uvedu alespoň stručný popis principu nejdůležitějších metod objektu mobota, a to metod zprostředkovávajících senzorická měření.

Mechanické nárazníky

Mechanické nárazníky jsou standardně rozmístěny po obvodu mobota. Při realizaci metod zprostředkovávajících měření pomocí těchto senzorů jsem vycházel z parametrů existujících reálných mobotů. Tito moboti mají tvar osmiúhelníku a ne každé z jejich hran jsou umístěny čtyři senzory. Jediným předpokladem, který jsem při realizaci metod mechanických nárazníků učinil, byl ten, že moboti budou mít tvar mnohoúhelníku. Objekt mobota je tedy z hlediska počtu hran, počtu senzorů na jedné hraně a samotné velikosti mobota zcela modifikovatelný. Parametry jimiž se tyto vlastnosti nastavují jsou dostupné z panelu parametrů mobota.

Senzory na jedné hraně jsou brány jako jediný signál, tzn. vyhodnocuje se pouze to, zda je některý z nich vybuzen, ale nezjišťuje se který. Senzory na jedné hraně se tedy jeví jako senzor jediný s kontaktními místy v místech jednotlivých senzorů. Počet hran mobota (*Number Of Bumpers*) je v tedy konečném důsledku totožný s počtem signálů přicházejících od nárazníků. Parametr citlivosti nárazníků (*Bumper's Quality*) není ničím jiným než počtem senzorů na jedné hraně, tedy počtem snímaných míst. Parametrem poloměru mobota (*Bumper's Outcircle Radius*) lze nastavit jeho velikost. Výsledný mnohoúhelník odpovídající tvaru mobota je vepsán do kružnice tohoto poloměru.

Metoda měření pomocí mechanických nárazníků tedy podle daných parametrů, aktuální pozice a úhlu natočení mobota, vypočte souřadnice, v nichž pak v obrázku prostředí testuje přítomnost jiné barvy než barvy podlahy (*Floor*) a příslušným způsobem nastaví hodnoty nárazníků (vybuzen/nevybuzen).

IR čidla a sonar

Existující moboti jsou vybaveni IR čidly a sonarem. Měření pomocí obou těchto senzorů je implementováno stejnými metodami. Jelikož sonar umožňuje měření vzdálenosti překážky v daném směru, je pomocí něj realizováno i měření IR čidly, které umožňují pouze zjistit, zda se v daném směru do vzdálenosti jejich dosahu nějaká překážka nachází. Jednoduše pak bude záležet jen na tom, zda bude brána v úvahu naměřená hodnota a nebo pouze aktivita daného senzoru. Dále tedy nebudu v textu mezi měřeními pomocí IR čidel a měřeními pomocí sonaru činit rozdíl.

Metody měření pomocí IR čidel jsou opět velmi flexibilní. Jejich výsledná funkce závisí na několika parametrech, které jsou opět dostupné z panelu parametrů mobota. Jsou to parametr počtu IR čidel (*Number Of IRs*), parametr jejich dosahu (*IR Range*) a parametr jejich rozlišení (*IR Step*).

Úhly, ve kterých měření probíhá jsou odvozeny z počtu IR čidel a aktuálního úhlu natočení mobota. V těchto úhlech jsou pak od okraje mobota až po zadaný dosah IR čidel vypočteny z příslušným krokem body, v nichž je opět se zahrnutím aktuální polohy mobota, testována přítomnost jiné barvy než barvy podlahy (*Floor*). Vzájemná vzdálenost jednotlivých bodů měření v daném úhlu (krok) odpovídá parametru rozlišení IR čidel (*IR Step*). Hodnoty IR čidel, jsou poté nastaveny v závislosti na tom, ve kterém kroku byla poprvé (měření probíhá směrem od obvodu mobota dál) zjištěna přítomnost překážky. První bod měření je ve vzdálenosti jednoho kroku měření (*IR Step*) od obrysu mobota, což znamená, že hodnoty IR čidel odpovídají vzdálenosti překážky od okraje mobota a nikoliv od jeho středu.

Senzory viditelnosti jiného mobota

Jak jsem již uvedl, jsou hodnoty senzorů viditelnosti jiného mobota vypočteny na základě vzájemné polohy mobotů. Informace o poloze druhého mobota jsou získány pomocí komunikace prostřednictvím tabule, jejíž mechanismus jsem již popsal.

Z aktuální pozice a úhlu natočení daného mobota a z aktuální pozice sledovaného mobota je vypočten úhel pod kterým daný mobot sledovaného mobota vidí. Tato hodnota je pak použita pro výpočet hodnot jednotlivých senzorů viditelnosti. Z důvodu znormování těchto hodnot jsou tyto počítány dle vztahu:

$$s_i = \begin{cases} \cos(\Phi_i) & if \quad \cos(\Phi_i) \geq 0 \\ 0 & jindy \end{cases},$$

kde

s_i … je hodnota,

Φ_i … je úhel, pod kterým mobot sledovaného mobota vidí právě vzhledem k poloze i-tého senzoru viditelnosti jiného mobota.

Hodnoty senzorů viditelnosti jiného mobota tedy budou tedy z intervalu <0,1>. Takto normované hodnoty již můžou být použity přímo jako vstupy ASN sítě. Je také jasné, že při měření bude nenulové hodnoty nabývat vždy větší počet zmíněných senzorů současně. To je ale z hlediska již dříve popsaného principu ASN výhodné.

Na počtu senzorů viditelnosti jiného mobota bude přímo záviset plynulost výsledného pohybu mobota při jeho cestě směrem k cíli. To je naprosto zřejmé, neboť počet těchto senzorů přímo udává rozlišovací schopnosti mobota, tj. počet úhlů pod nimiž je tento mobot schopen přítomnost jiného mobota zaregistrovat.

6.3.2 Dosažení statického cíle

V tomto odstavci uvedu výsledky pokusů, které jsem po vytvoření simulátoru a základních tříd objektů provedl.

Nejprve jsem odladil samotnou funkčnost všech použitých metod a algoritmů. K tomuto účelu jsem vytvořil třídu TSimpleMobotThread, jejíž popis a odpovídající trajektorii pohybu mobota jsem již uvedl v odstavci 6.3.1.3 - *Objekt Mobota*.

Poté jsem provedl sérii testů, při nichž jsem zjišťoval především vliv parametrů ASN na výsledný pohyb mobota. Snažil jsem se najít určité optimální nastavení, při kterém by průběh učení a především výsledná trajektorie pohybu mobota byly co nejlepší.

Nastavení parametrů ASN jsem provedl spíše intuitivně, než na základě přesně specifikovaného postupu. Jednotlivé parametry jsem odladil na základě mnoha pokusů, při nichž jsem sledoval vývoj hodnot synaptických vah neuronů ASN a samotný pohyb mobota. Parametry jsem v konečné fázi testů nastavil tak, aby především v počáteční fázi učení nedocházelo k výraznému odlišení hodnot některých vah, což mělo za následek převážení příslušných výstupů nad ostatními a k uvíznutí ASN v daném stavu, jelikož tak již ostatním výstupům nebylo umožněno se dále projevit. Pokud nedošlo přímo k uvíznutí, docházelo tímto způsobem k nežádoucím situacím, kdy mobot začal preferovat určitý směr svého pohybu a dosažení cílové pozice tak bylo ve většině případů zcela znemožněno. Dále jsem se soustředil na to, aby náhodná složka podílející se na hodnotách výstupů ASN byla natolik velká, aby se nepřestala uplatňovat

již po několika učících krocích ASN, což by také vedlo k počátečnímu preferování určitého pohybu, a současně aby byla natolik malá, aby se při určité úrovni znalostí ASN projevovat naopak přestala a nenarušovala dále již dostatečně efektivní trajektorii pohybu mobota. Tímto postupem nalezené hodnoty jsou: konstanta rychlosti učení $\eta = 0.25$, střední hodnota náhodné veličiny 0.5 a rozptyl 0.01, konstanta rychlosti učení pro nultý vstup $\eta_0 = 0.5$, omezení vah odpovídajících nultému vstupu $MEZ = 0.005$. Při tomto nastavení jsem posléze prováděl všechny následující pokusy, a tudíž tyto hodnoty již dále nebudu znovu uvádět.

Hlavními oblastmi na které jsem se při dalších testech soustředil byl způsob korekce vah neuronů a vliv počtu senzorů viditelnosti jiného mobota na výslednou trajektorii pohybu.

6.3.2.1 Vliv různých způsobů korekce vah neuronů

Korekci synaptických vah neuronů ASN, tj. učení sítě, lze provádět na základě dvou přístupů. Při prvním z nich dochází k odpovídajícím úpravám hodnot vah pouze při změně hodnoty příslušného výstupu neuronu v předchozím kroku, tzn. jeho nastavení na 1 (aktivaci) a nastavení na 0 (deaktivaci). Jinými slovy jsou tak sledovány pouze vlivy změn výstupů neuronů na změny v hodnotě signálu *payoff*. V druhém případě dochází k odpovídajícím korekcím v případě, kdy byla v předcházejícím kroku hodnota příslušného výstupu rovna jedné, tzn. i v případě že ke změně hodnoty tohoto vstupu nedošlo. Jinými slovy je tak u jednotlivých neuronů sledováno přímé působení výstupů na změny v hodnotě *payoff*.

V prvním případě jsou tedy hodnoty vah upravovány dle vztahu:

$$w_{ij}(t+1) = w_{ij}(t) + \eta[z(t) - z(t-1)] \cdot [y_j(t-1) - y_j(t-2)] x_{ij}(t-1)$$

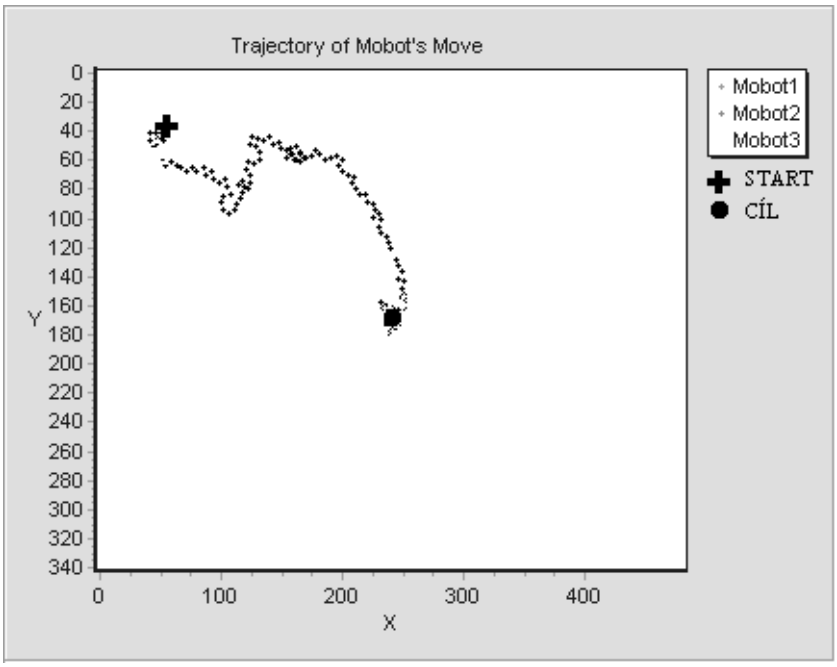
a ve druhém případě pak dle vztahu:

$$w_{ij}(t+1) = w_{ij}(t) + \eta[z(t) - z(t-1)] y_j(t-1) x_i(t-1),$$

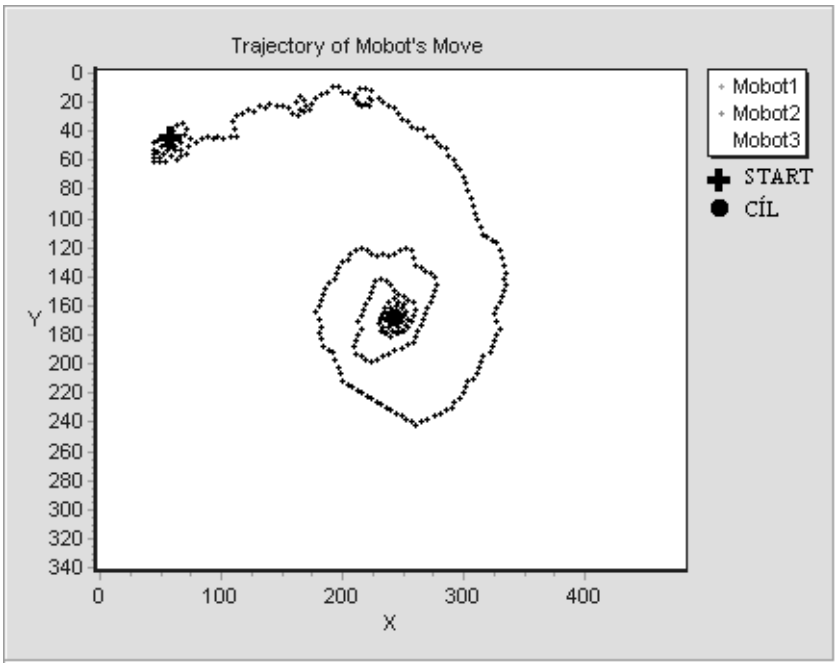
kde významy jednotlivých členů a podrobnější diskuse rozdílu těchto vztahů je uvedena v odstavcích 6.1.2 - *ASN* a 6.1.3 - *Vrstva koordinace*.

Trajektorie pohybu mobota při korekcích vah dle prvního přístupu, tj. sledováním změn výstupů příslušných neuronů, je uvedena na obr. 31. Trajektorie pohybu v případě korekce vah použitím druhého přístupu, tj. přímým působením výstupů, je uvedena na obr. 32. Z těchto obrázků je naprosto patrný obrovský přínos sledování změn výstupů jednotlivých neuronů, tj. úpravy vah pouze těch neuronů, u nichž došlo v minulém kroku ke změně v hodnotě výstupu. To odpovídá logickým předpokladům, jelikož při úpravách vah i v situacích, kdy sice hodnota výstupu příslušného neuronu v předcházejícím kroku byla rovna jedné, ale při přechodu do tohoto stavu nedošlo k vlastní změně hodnoty tohoto výstupu, dochází k určité dezinformaci neuronové sítě. Podle vývoje *payoff* signálu v posledním kroku jsou tak totiž upravovány i váhy

neuronů, jež se na akci, která danou změnu v signálu *payoff* způsobila, nepodílely přímo. To je příčinou výrazně horšího způsobu pohybu mobota a tedy výsledné trajektorie jeho pohybu směrem k cíli.



obr. 31 - Trajektorie pohybu – korekce vah při změnách výstupů neuronů



obr. 32 - Trajektorie pohybu – korekce vah při aktivních výstupech neuronů

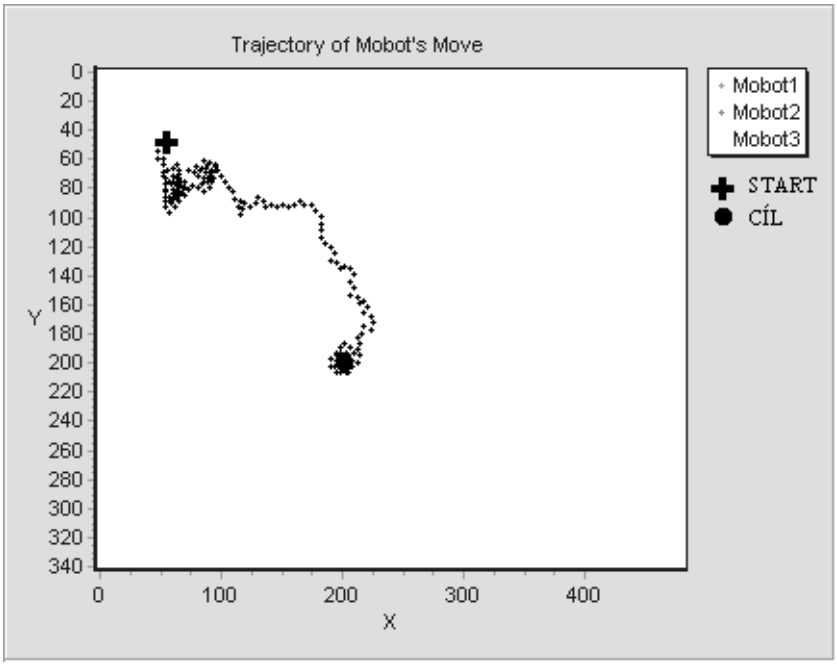
Na základě těchto výsledků budu tedy při dalších pokusech používat pro učení ASN sítě první z uvedených přístupů, tj. korekci hodnot vah pouze při změnách hodnot výstupů příslušných neuronů v minulém kroku.

6.3.2.2 Vliv počtu senzorů viditelnosti jiného mobota

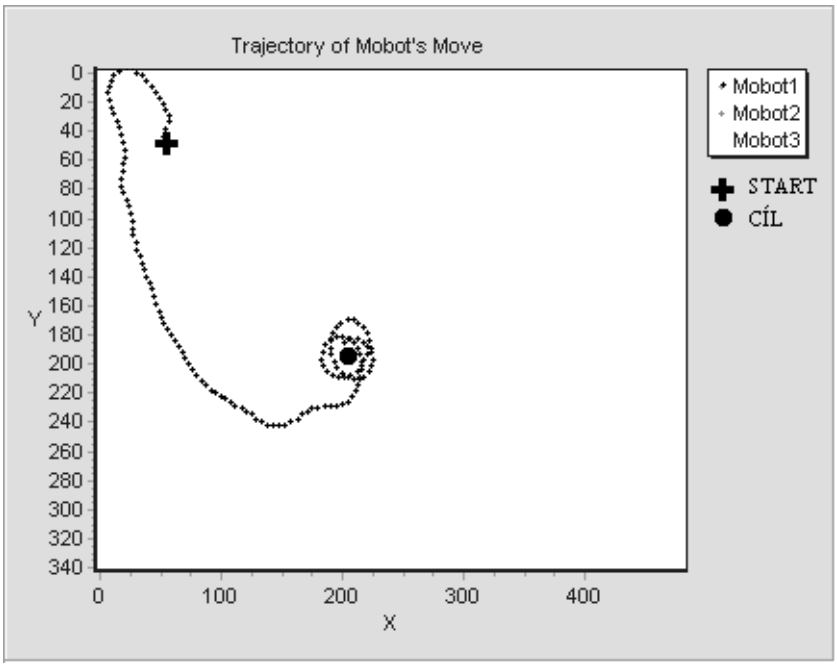
Jak jsem uvedl v odstavci 6.3.1.3 - *Objekt Mobota*, jednotlivými vstupy ASN sítě (kontextem) jsou normované hodnoty senzorů viditelnosti jiného mobota (resp. cíle), jejichž princip jsem objasnil v tomtéž odstavci. Na počtu senzorů viditelnosti jiného mobota bude tedy přímo záviset plynulost výsledného pohybu mobota při jeho cestě směrem k cíli. To je naprosto zřejmé, neboť počet těchto senzorů přímo udává rozlišovací schopnosti mobota, tj. počet úhlů pod nimiž je tento mobot schopen přítomnost jiného mobota zaregistrovat.

V této oblasti jsem provedl několik testů, z jejichž výsledků jsem zjistil, že samotné zvýšení počtu zmíněných senzorů mobota, což odpovídá zvýšení počtu vstupů ASN, však k podstatnému zlepšení plynulosti jeho pohybu nestačí. Tímto způsobem je sice mobotovi zprostředkována mnohem přesnější informace o směru, ve kterém jiného mobota (cíl) vidí, nicméně počet možných směrů jeho dalšího pohybu, odpovídajících počtu výstupů ASN sítě, zůstane stejný. Z tohoto důvodu tedy sice trajektorie pohybu směřuje k cíli přesněji, ale z hlediska plynulosti se její vlastnosti nezmění.

Na základě těchto poznatků jsem tedy provedl několik testů, při nichž jsem příslušným způsobem měnil s počtem vstupů ASN i počet jejích výstupů. Zjistil jsem, že optimální chování síť vykazuje při srovnatelném počtu vstupů a výstupů. Při vlastní reprezentaci prostředí obrázkem ve formě bitmapy jsem zvolil rozlišení cm/pixel. K této hodnotě jsem dospěl z minimálního kroku mobota. Z hlediska přesnosti dosažitelné v reálném prostředí jsem minimální krok mobota zvolil 5cm. To při daném rozlišení umožňuje z daného bodu pohyb do 40-ti různých směrů, což znamená minimální možný úhel natočení mobota 9°. Z hlediska přesnosti reálných mobotů je i tato hodnota více než vyhovující. Z uvedeného tedy vyplývá omezení maximálního počtu výstupů ASN sítě na 40. Výsledky testů vlivu počtu senzorů viditelnosti jiného mobota na plynulost výsledného pohybu mobota jsou patrné z obr. 33 a obr. 34. Na obr. 33 je uvedena trajektorie pohybu mobota se 4-mi senzory viditelnosti jiného mobota a tedy se 4-mi výstupy ASN. Na obr. 34 je uvedena trajektorie pohybu mobota se 40-ti senzory viditelnosti a tedy 40-ti výstupy ASN (tj. počet požadovaných směrů pohybu 40).

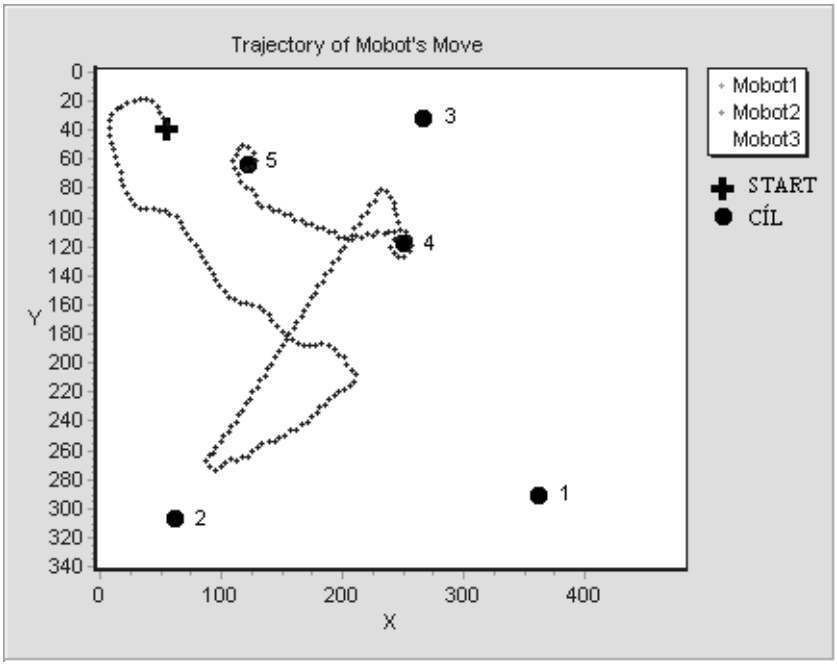


obr. 33 - Trajektorie pohybu při počtu vstupů a výstupů ASN 4x4



obr. 34 - Trajektorie pohybu při počtu vstupů a výstupů ASN 40x40

Použití většího počtu senzorů viditelnosti jiného mobota je tedy z hlediska plynulosti pohybu značným přínosem a proto bude v dalších simulacích používán mobot vybavený 40-ti senzory viditelnosti jiného mobota (40-ti vstupy ASN) a se 40-ti možnými požadovanými směry pohybu (40-ti výstupy ASN). Tohoto mobota jsem otestoval ještě z pohledu dynamičnosti jeho reakcí na změny polohy cíle (pozice 1-5 na obr. 35) v okamžicích, kdy je mobot již v pohybu. Z výsledné trajektorie pohybu mobota, uvedené na obr. 35, jsou vidět více než uspokojivé výsledky, tj. rychlá reakce mobota při zachování plynulosti pohybu.

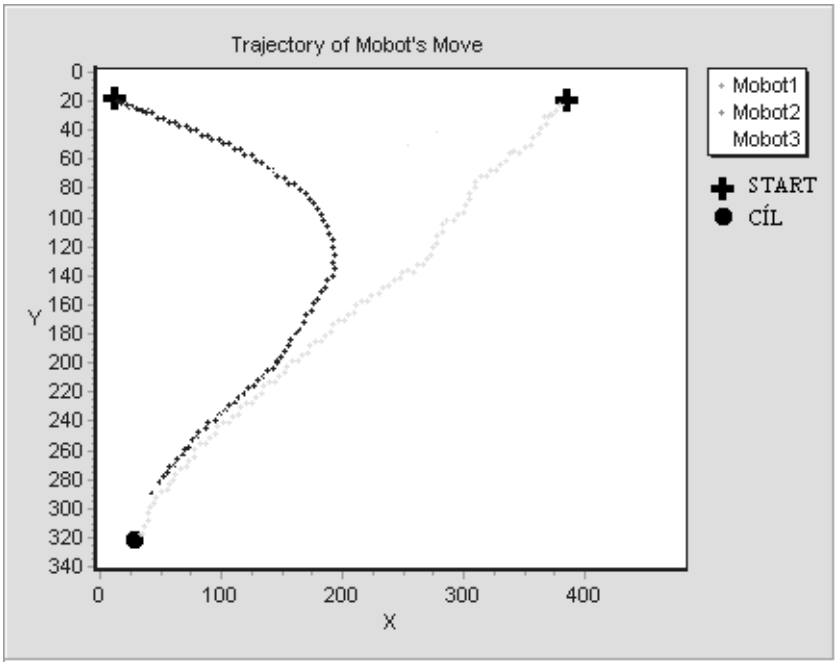


obr. 35 - Trajektorie pohybu při náhlých změnách cíle (1-5)

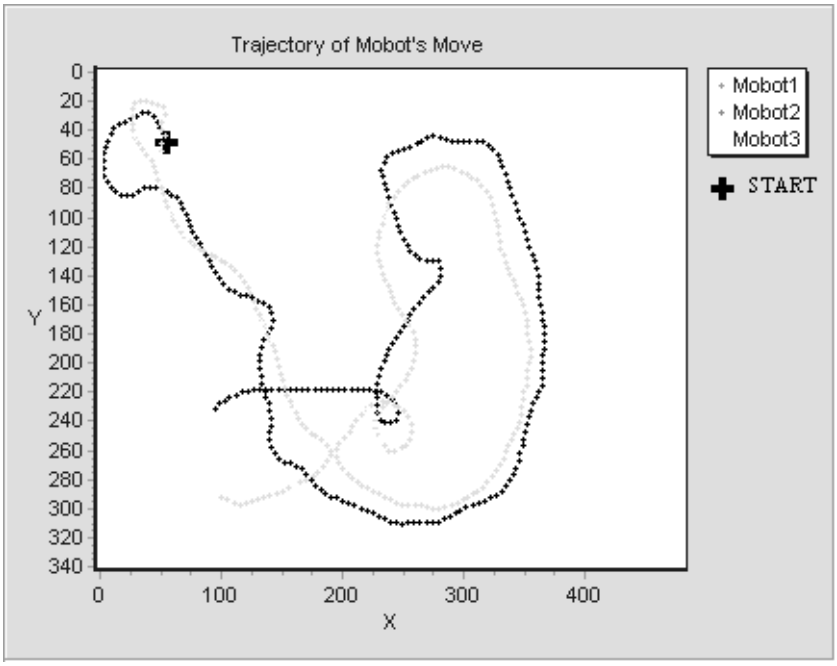
6.3.3 Stíhání dynamického cíle

V předcházející etapě jsem tedy postupně odladil všechny potřebné parametry mobota (objektu simulátoru) a na základě provedených simulací dospěl až k jeho konečné verzi, a to se 40-ti senzory viditelnosti jiného mobota a stejným počtem možných požadovaných směrů dalšího pohybu (40-ti výstupy ASN).

Nezbývá než ukázat koordinaci pohybu jednoho mobota s pohybem mobota jiného, tj. ukázat chování při dynamicky se měnící poloze cíle. Pro demonstraci koordinace pohybu uvádím dva příklady. V prvním z nich, uvedeném na obr. 36, má jeden z mobotů startovní pozici v pravém horním rohu obrázku a pohybuje se směrem ke statickému cíli vlevo dole. Druhý mobot má za úkol stíhat mobota prvního a startovní pozici má v levém horním rohu obrázku. Pohyb obou mobotů byl při této simulaci spuštěn ve stejný okamžik. Ve druhém příkladu, uvedeném na obr. 37, je pak již uvedena úplná vzájemná koordinace obou mobotů. První mobot má za úkol stíhat mobota druhého, který naopak před prvním mobotem "prchá". Zde chci podotknout, že tohoto odlišného chování je docíleno pouze změnou hodnotícího signálu dodávaného druhému mobotovi a nikoliv zásahem do architektury řídícího systému. To demonstruje flexibilitu přístupu z oblasti posilovaného učení (Reinforcement learning), jakým je řešení prostřednictvím asociativně vyhledávací neuronové sítě (ASN). Druhému mobotovi je prostě jen dodáván hodnotící signál inverzní vzhledem k signálu dodávanému mobotovi prvnímu. To je vše, co je pro odlišení jejich výsledného způsobu koordinace pohybu potřeba zajistit. Jednoduchým způsobem by tak bylo možno zajistit další úlohy, při kterých by se jeden mobot například mohl snažit k druhému mobotovi přiblížit pouze na určitou vzdálenost, případně udržovat vůči němu tuto vzdálenost pod určitým úhlem apod. V podstatě lze realizovat ty úlohy, ke kterým jsme schopni generovat odpovídající hodnotící signál. Problémy nastanou v případě, kdy bude mít hodnotící signál lokální maxima a kdy bude hrozit uvíznutí v některém z nich. Tím se ale v rámci této práce zabývat nebudu.



obr. 36 - Stíhání dynamického cíle



obr. 37 - Vzájemná koordinace pohybu dvou mobotů

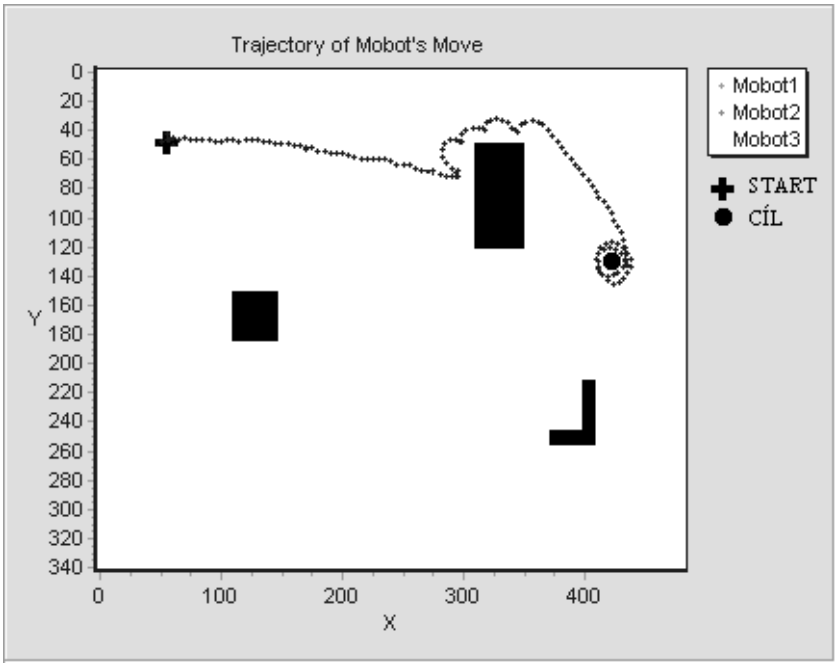
Je tedy vidět, že docílení vzájemné koordinace pohybu dvou mobotů prostřednictvím navrženého řídícího systému a implementací vrstvy koordinace pomocí asociativně vyhledávací neuronové sítě, se podařilo s dostatečně uspokojujícími výsledky. Vzájemná koordinace pohybu mobotů, jež byla hlavním cílem této práce, proběhla v uvedených testech nad očekávání dobře.

6.3.4 Rozšíření o základní reflexní vazby

V této poslední etapě jsem rozšířil řídící algoritmus mobota o nejnižší úroveň reflexních vazeb, abych tak demonstroval funkčnost řídícího systému při implementaci další z jeho vrstev.

Jelikož předvedení funkčnosti řízení při realizaci dvou vrstev bylo jediným záměrem, vytvořil jsem vrstvu reflexních vazeb nejjednodušším možným způsobem. Při nárazu na překážku převezme řízení pohybu mobota právě zmíněná vrstva reflexních vazeb. Tato nejprve zajistí couvnutí mobota o vzdálenost nutnou k umožnění jeho bezpečného otočení a poté provede vlastní otočení mobota o přesně definovaný úhel, tj. o 90° doleva. Nakonec zajistí provedení dalšího kroku mobota. Pokud v tomto kroku již k nárazu nedojde, převezme řízení opět modul koordinace, který se opět začne snažit nasměrovat pohyb mobota směrem k cíli.

Na obr. 38 jsem uvedl výslednou trajektorii mobota při jednom z testů, které jsem v prostředí obsahujícím překážky provedl. Z trajektorie pohybu mobota je vidět nedokonalost implementace vrstvy reflexních vazeb. Při natočení mobota a jeho vzdálení se od překážky je předáno řízení modulu koordinace příliš brzy, a to vede k několikanásobnému opětovnému nárazu. Správná funkčnost řízení při implementaci dvou vrstev je však z obrázku zcela patrná.



obr. 38 - Trajektorie pohybu při implementaci vrstvy reflexních vazeb

6.4 Možné další směry vývoje

V tomto odstavci uvádím některé z dalších možných cest vývoje řídícího systému mobota, které jsou logickým posunem v dosaženém stupni realizace, ke kterým jsem směřoval a kterým bych se chtěl dále věnovat v rámci svého doktorandského studia.

6.4.1 Modul vyhýbání se překážkám

Zcela zásadní věcí je rozšíření o vrstvu zajišťující vyhýbání se překážkám. Tato by měla plně využívat informací z IR čidel, popřípadě sonaru, a koordinovat tak trajektorii mobota v blízkosti překážek.

Zde chci podotknout, že jsem v této oblasti již podnikl několik pokusů, a to sice v tom směru, že jsem se pokusil informace z IR čidel přidat přímo jako další vstupy do ASN sítě realizující koordinaci pohybu. Těmto vstupům jsem nastavil větší rozsah hodnot, což zajistilo jejich vyšší prioritu před vstupy viditelnosti druhého robota. Náraz na překážku zajistil snížení *payoff* signálu, které bylo informací o špatném charakteru poslední akce. Jelikož IR čidla měly vyšší hodnoty, zapsala se informace do neuronové sítě především prostřednictvím synaptických vah odpovídajících těmto vstupům, což vyplývá ze skutečnosti, že vztah pro korekci vah obsahuje člen hodnoty vstupu v minulém kroku. Tímto způsobem se při zareagování IR čidel, měly projevit především jim odpovídající vstupy neuronů a zajistit tak vyhnutí se nárazu s překážkou. Jak jsem se ale přesvědčil několika pokusy, nastavit správný vzájemný poměr maximálních hodnot vstupů IR čidel a senzorů viditelnosti druhého robota, je zřejmě nemožné. Navíc úpravy vah IR čidel při nárazu se nevhodně projevovaly i na vahách odpovídajících vstupům senzorů viditelnosti druhého mobota a znehodnocovaly tak informaci, kterou se ASN již naučila. Poslední a nejpodstatnější neblahou skutečností bylo to, že se mobot často dostal do stavu, kdy ASN síť přestala generovat jakýkoli výstup, jelikož právě váhy vstupů odpovídajících IR čidlům potlačily váhy odpovídající vstupům senzorů

viditelnosti druhého robota požadující další pohyb. Jelikož tak nebyla vygenerována další akce, tedy požadavek na otočení, zůstal mobot stát, IR čidla vybuzeny a mobot tím pádem tento stav již nikdy nemohl opusit. Z těchto důvodů jsem tento způsob řešení vyhýbání se překážkám, tedy prostřednictvím rozšíření ASN sítě řídící koordinaci, zavrhl.

Vyhýbání se překážkám by tedy mělo být realizováno vlastní vrstvou, řídícího systému, tak jak jsem uvedl již při popisu globální struktury na obr. 21. Tato by mohla být složena z několika modulů, řídící různé způsoby vyhýbání se překážkám. Jeden z nejjednodušších modulů by mohl při nárazu na překážku zajišťovat otočení robota a sledování okraje překážky do doby, kdy by požadavek směru dalšího pohybu vrstvou koordinace, či kteroukoli jinou z vyšších vrstev, již nebyl příčinou opětovného nárazu na překážku. Složitější modul této vrstvy, umožňující efektivnější způsob objíždění překážek, by mohl být realizován opět ASN sítí, pro kterou by vstupy byly údaje IR čidel, či sonaru, měřících vzdálenost od překážky, a tato vzdálenost by byla současně použita jako hodnotící signál. Tuto vzdálenost by se tedy ASN snažila maximalizovat, a tudíž by se tak korigoval výsledný směr pohybu mimo překážku. Konflikty mezi požadavky této vrstvy a ostatními vrstvami řídícího systému mobota, jakou je např. právě vrstva koordinace, by byly jednoduše řešeny na úrovni modulu správy akcí, tak jak jsem popsal v odstavci 6.1.1.1 - *Modul správy akcí*.

6.4.2 Koordinace vzájemné polohy více mobotů

Jako zásadní krok kupředu v rozvoji řídícího systému mobota pak vidím rozšíření vrstvy koordinace, a to ve smyslu umožnění stíhání jednoho robota skupinou jiných. Prozatím je vytvořená vrstva, resp. modul koordinace schopen zajišťovat stíhání jednoho robota, tedy koordinovat vůči němu svoji polohu, což sice umožňuje realizovat úlohu, kdy více robotů stíhá jednoho, ale tito tak činí bez koordinace pohybu mezi sebou samými.

Koordinace vzájemné polohy mezi skupinou mobotů, by mohla být buďto zajištěna přímým rozšířením ASN modulu koordinace, a nebo opět samostatným modulem. Jelikož by se rozšíření ASN týkalo pouze o zavedení dalších vstupů přinášejících informace o viditelnosti dalších členů skupiny, mezi nimiž by nebyl činěn rozdíl v jejich prioritách, nemuselo by toto rozšíření přinést problémy naznačené v předchozím odstavci 6.4.1. Zde jsou ale zřejmé dva možné přístupy. Jde o to, zda při lovu ve skupině má být kladen větší důraz na držení určité formace lovců, a nebo na co nejrychlejším přiblížení se k lovenému objektu. První možnost je výhodnější například při počátečním hledání loveného objektu prohledáváním co největší oblasti a v prvních momentech zahájení lovu, kdy je tak, např. tzv. rojnicí, znemožněno třeba rychlejšímu lovenému objektu vyváznout. Druhá možnost, tedy větší priorita přiblížení se k cíli, je vhodná v následující fázi lovu, kdy je již úpravou formace lovený objekt určitým způsobem omezen ve svém pohybu, nicméně kdy je nutné již mnohem dynamičtěji reagovat právě na změny jeho polohy.

Pro zachování možností obou těchto přístupů, je tedy dle mého názoru vhodnější provést rozšíření schopností vrstvy koordinace vytvořením jejího dalšího modulu. To nám totiž umožní v daných situacích příslušným způsobem upravovat, na úrovni modulu správy akcí, vliv obou těchto zmíněných strategií a docílit tak vyšší efektivity výsledného chování.

Jednou z možností je tento modul vytvořit opět pomocí asociativně vyhledávací sítě, jejímiž vstupy budou informace o viditelnosti ostatních členů skupiny mobotů, přičemž hodnotící signál *payoff*, bude určitým způsobem svázán z podobností současné formace s požadovanou. Jako příklad zde vidím *payoff* signál, který při velké

vzdálenosti od loveného objektu nabývá svého maxima v případě, že mobot vidí jednoho ze svých kolegů přesně napravo od sebe a druhého přesně nalevo, tzn. pod úhly ± 90°. Při přibližování se k cíli by se pak toto jeho maximum pomalu přesouvalo směrem k situaci, kdy by své dva kolegy viděl například pod úhly ± 60°, což by mělo za následek postupné svírání formace a tím obklíčení loveného objektu. Zde však musím podotknout, že jde jen o určité úvahy a předpoklady, a ty se, jak jsem se mnohokrát při tvorbě této práce přesvědčil, nakonec mnohdy velmi výrazně liší od reálných výsledků.

7 Závěr

V průběhu této práce jsem navrhl řídící systém mobilního robota (mobota), umožňující koordinaci jeho jednání s jiným mobotem, popř. v rámci skupiny. Předchozí prostudování základních principů chování živých organismů z pohledu etologie samotné a základních přístupů a metod distribuované umělé inteligence, mne vedlo k použití reaktivního, chováním motivovaného způsobu řízení. Dále jsem implementoval část tohoto řídícího systému umožňující vzájemnou koordinaci pohybu dvou mobotů. Jako prostředku k její realizaci jsem použil asociativně vyhledávací neuronovou síť (Associative Search Network).

Podstatnou součástí práce bylo navržení demonstrační úlohy a především vytvoření programu simulátoru, pomocí něhož jsem funkčnost řídícího algoritmu ověřil v konečné fázi sérií testů. Většina testů proběhla nad očekávání úspěšně. Mobot se pohyboval zcela plynule, bez zbytečných odchylek od přímé trajektorie směrem k cíli a také dynamičnost jeho reakcí na změny polohy cíle byla více než uspokojující. Tyto dosažené vlastnosti jsem poté demonstroval na konečné úloze vzájemné koordinace pohybu dvou mobotů, kdy měl první mobot za úkol stíhat mobota druhého, který se naopak před prvním mobotem snažil utéct. Samotný výběr použitého přístupu, tj. tzv. posilovaného učení (Reinforcement learning), který neuronová síť ASN využívá, se z hlediska dosáhnutých výsledků potvrdil jako velmi vhodný. Především se prokázala jeho značná flexibilita v možné modifikaci výsledné funkce chování v závislosti na poskytnutém hodnotícím signálu. Toho jsem využil především u realizace konečné úlohy, kdy jsem odlišného chování druhého robota, tzn. jeho snahu vzdálit se od mobota prvního, realizoval pouze tím, že jsem mu dodával hodnotící signál inverzní, vzhledem k signálu dodávanému prvnímu mobotovi.

Při návrhu řídícího systému jsem se v maximální míře soustředil na umožnění pozdější realizace pomocí již existujících mobotů, jimiž disponuje mobotická skupina na katedře kybernetiky elektrotechnické fakulty ČVUT v Praze. Z důvodu časové náročnosti jejich nutného doplnění o některé další zdroje informace, jsem však, na pokyn vedoucího práce, od oblasti implementace úlohy na reálných robotech ustoupil. Zmíněné oblasti a stejně tak dalšímu vývoji samotného řídícího systému bych se chtěl věnovat v rámci svého doktorandského studia.

Použitá literatura

[Barto, 1981] Barto A. G., Sutton R. S., Brouwer P. S.: Associative Search Network: A Reinforcement Learning Associative Memory. Biological Cybernetics, Springer Verlag, 1981, vol. 40, str. 201-211.

[Barto(2), 1981] Barto A. G., Sutton R. S.: Landmark Learning: An Illustration of Associative Search. Biological Cybernetics, Springer Verlag, 1981, vol. 42, str. 1-8.

[Barto, 1983] Barto A. G., Sutton R. S., Anderson Ch. W.: Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man, and Cybernetics, vol. smc-13, no. 5, September/October 1983, str. 834-841.

[Barnes, 1997] Barnes D. P., Ghanea-Hercock R. A., Aylett R. S., Coddington A. M.: Many hands make light work? An investigation into behaviourally controlled co-operant autonomous mobile robots. Proceedings of the first International Conference on Autonomous Agents, February 5-8 (1997), str. 413-420.

[Blum, 1992] Blum A.: Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems. John Wiley & Sons, Inc., New York, 1992.

[Buchtelová, 1998] Buchtelová R., Confortiová H., Červená V., a kolektiv: Akademický slovník cizích slov. Academia, Praha, 1998, str. 207.

[Ferber, 1996] Ferber J.: Reactive Distributed Artificial Intelligence: Principles and Applications. Foundations of Distributed Artificial Intelligence (G. M. P. O'Hare and N. R. Jennings), John Wiley & Sons, Inc., New York, 1996, str. 287-314.

[Franck, 1996] Franck D.: Etologie. Karolinum, vydavatelství Univerzity Karlovy, Praha, 1996.

[Freeman, 1991] Freeman J. A., Skapura D. M.: Neural Networks: Algorithms, Applications, and Programming Techniques. Addison-Wesley, New York, 1991.

[Gasser, 1991] Gasser L.: Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. Artificial Intelligence, vol. 47 (1991), str. 107-138.

[Haykin, 1999] Haykin S.: Neural Networks: a Comprehensive Foundation. Prentice Hall: Upper Saddle River, New Jersey, 1996.

[Hebb, 1949] Hebb D.: The Organization of Behavior. John Wiley & Sons, Inc., New York, 1949.

[Hopfield, 1982] Hopfield J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, vol. 79 (1982), str. 2445-2558.

[Jiřina, 1995] Jiřina M.: Neuro-fuzzy algoritmy. Diplomová práce, ČVUT FEL, 1995.

[Kröse, 1998] Kröse B.: Environment learning and localization in "sensor space". Proc. of the Tenth Netherlands/Belgium Conf. on Artificial Intelligence, November 1998, str. 229-239.

[Literák, 1999] Literák L.: Neuronové sítě. Diplomová práce, Přírodovědecká fakulta Univerzity Palackého, katedra matematické informatiky, 1999, str. 1-16.

[Lorenz, 1993] Lorenz K.: Základy etologie: Srovnávací výzkum chování. Academia, Praha, 1993.

[Matarić, 1998] Matarić M. J.: Coordination and Learning in Multi-Robot Systems. IEEE Intelligent Systems, March/April 1998, str. 6-8.

[Matarić, 1995] Matarić M. J.: Issues and Approaches in the Design of Collective Autonomous Agents. Robotics and Autonomous Systems, 16(2-4), December 1995, str. 321-331.

[Matarić, 1997] Matarić M. J.: Reinforcement Learning in the Multi-Robot Domain. Autonomous Robots, 4(1), March 1997, str. 73-83.

[Mařík, 1997] Mařík V., Štěpánková O., Lažanský J., a kolektiv: Umělá inteligence (2). Academia, Praha, 1997, str. 142-177.

[Masters, 1995] Masters T.: Advanced Algorithms for Neural Networks: A C++ Sourcebook. John Wiley & Sons, Inc., New York, 1995.

[McCulloch, 1943] McCulloch W. S., Pitts W.: A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, vol. 5 (1943), str. 127-147.

[Minsky, 1969] Minsky M., Papert S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA, 1969.

[Rosenblatt, 1958] Rosenblatt F.: The perceptron: A Probablistic Model for Information Storage and Organization in the Brain. Psychological review, vol. 65 (1958), str. 386-408.

[Rosenblatt, 1962] Rosenblatt F.: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington, D.C., 1962.

[Rumelhart, 1986] Rumelhart D. E., Hinton G. E., and Williams R. J.: Learning Internal Representations by Error Propagation. Parallel Distributed Processing, vol. 1 (1986), MIT Press, Cambridge, MA, str. 318-362.

[Shepherd, 1995] Sheppard M., Oswald A., Valenzuela C., Sullivan G., Sotudeh R.: Reinforcement Learning in Control. University of Teesside, Middlesbrough, Cleveland, 1995.

[Šnorek, 1996] Šnorek M., Jiřina M.: Neuronové sítě a neuropočítače. ČVUT, Praha, 1996.

[Tinbergen, 1951] Tinbergen N.: The Study of Instinct. 1. Aufl. Oxford Univ. Press, 1951.

[Wooldridge, 1995] Wooldridge M. J., Jennings N. R. (eds.): Intelligent Agents. LNAI No. 890, Springer Verlag, Heidelberg, 1995.

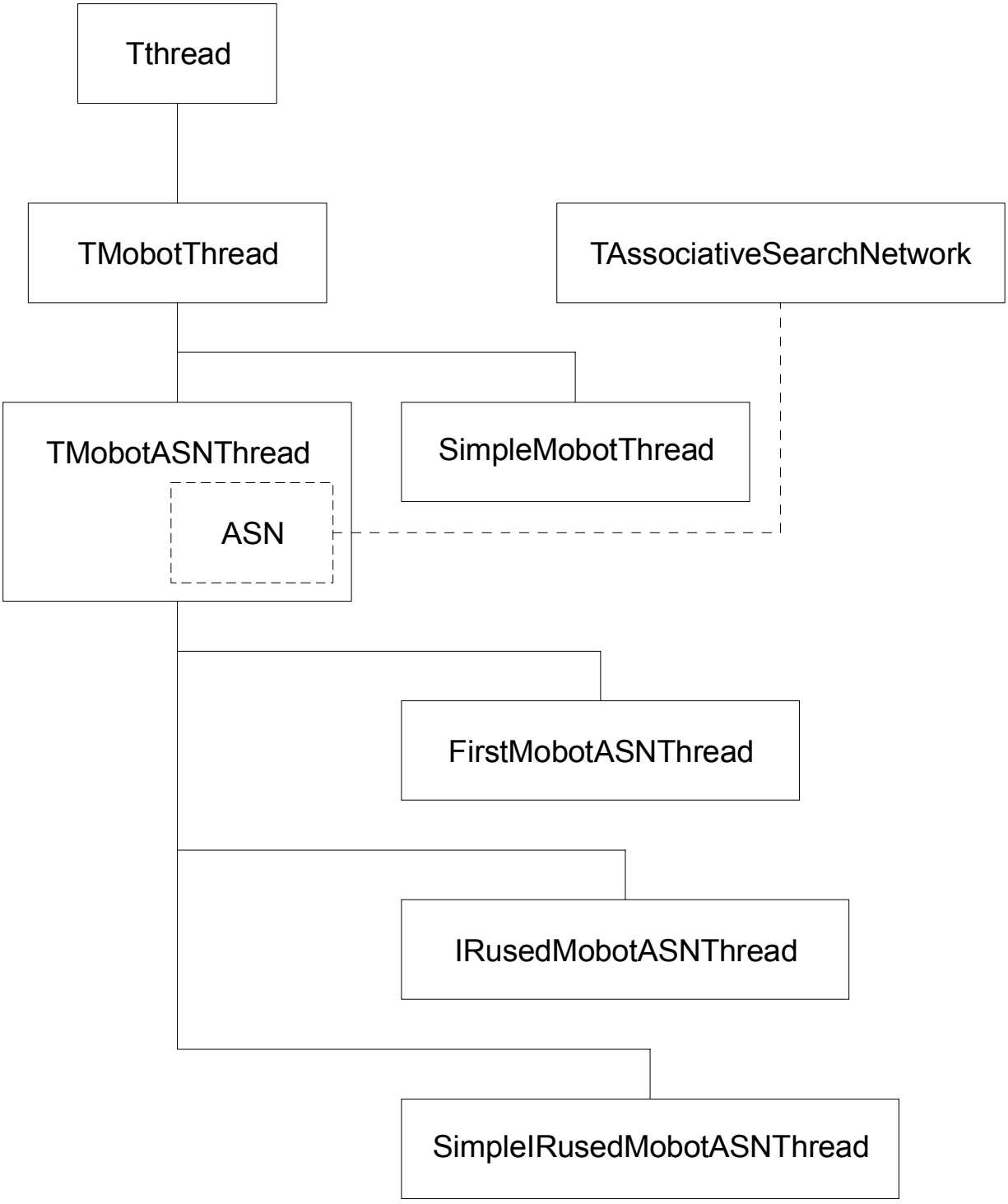
[URL, Garforth] http://www.janus.demon.co.uk/alife/notes/subsump.html, Jason P. Garforth: Subsumption Architecture, Artificial Life and Robotics, 1996.

[URL, University of Michigan] http://ai.eecs.umich.edu/cogarch0/subsump/index.html, A Survey of Cognitive and Agent Architectures, University of Michigan, 1994.

[URL, University of Stuttgart] http://www.gm.fh-koeln.de/~west/snns/index.htm, Zell A., Mamier G., Vogt M. et al.: Stuttgart Neural Network Simulator, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1995.

[URL, University of Southern California] http://www-robotics.usc.edu/~monica, Nicolescu M.: Learning the foraging task in o multi-robot domain. University of Southern California, 1998.

Příloha 1: Vztahy mezi objekty



Příloha 2: Definice třídy TMobotThread

```
//---------------------------------------------------------------------------
#ifndef ClassMobotThreadH
#define ClassMobotThreadH
//---------------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
class TMobotThread : public TThread
{
private:

   // only fields
   bool FParametersChangeRequest;
   int FLastCoordinateX;
   int FLastCoordinateY;
   float FLastDirection;
   int FCollisionStepBackSize;
   float FBumpersAngle;
   float FBumpersHalfAngle;
   float FIRsAngle;
   float FIRsHalfAngle;
   int FMobotBorderOutCircleRadius;

   // protected properties with direct access ( readonly )
   int *FIRState;
   int *FBumperState;
   bool FSomeIRIsActive;
   bool FSomeBumperIsActive;
   float FMobotTurnAngleStep;
   int FMobotTurnAngleStepsToPI;
   // protected properties with set and get methods
   int FCoordinateX;
   int FCoordinateY;
   float FDirection;

   // public properties with direct access ( readonly )
   int **FTrajectory;
   // public properties with direct access
   bool FPresenceInEnvironment;
   bool FNotAlone;
   bool FControlPanelLights;
   bool FControlPanelIRMeasuredValues;
   bool FControlPanelOnlyLastActivation;
   bool FTrajectoryRecord;
   TColor FColorOfMobot;
   TColor FColorOfMobotID;
   TColor FColorOfFloor;
   TColor FColorOfActivatedSensors;
   TColor FColorOfInActivatedSensors;
   // public properties with set and get methods
   int FMobotID;
   int FMobotMoveStep;
   int FNumberOfIRs;
   int FIRRange;
```

```
    int FIRStep;
    int FNumberOfBumpers;
    int FBumperOutCircleRadius;
    int FBumperQuality;
    int FNumberOfTrajectoryPoints;
    TPaintBox *FPaintBox;
    TImage *FEnvironment;
    int FInitialCoordinateX;
    int FInitialCoordinateY;
    int FInitialDirection;


    void __fastcall IRStateAllocateAndInitialize();
    void __fastcall IRStateDeAllocate();
    void __fastcall BumperStateAllocateAndInitialize();
    void __fastcall BumperStateDeAllocate();
    void __fastcall TrajectoryAllocateAndInitialize();
    void __fastcall TrajectoryDeAllocate();
    void __fastcall CollisionStepBackSizeInitialize();
    void __fastcall BumpersAngleInitialize();
    void __fastcall BumpersHalfAngleInitialize();
    void __fastcall IRsAngleInitialize();
    void __fastcall IRsHalfAngleInitialize();
    void __fastcall MobotBorderOutCircleRadiusInitialize();
    void __fastcall MobotTurnAngleStepInitialize();

    void __fastcall SetCoordinateX(int value);
    int __fastcall GetCoordinateX();
    void __fastcall SetCoordinateY(int value);
    int __fastcall GetCoordinateY();
    void __fastcall SetDirection(float value);
    float __fastcall GetDirection();

    void __fastcall AddPointToTrajectory();  // Add Point(FCoordinateX,FCoordinateY) to FTrajectory
array
    void __fastcall DoPaint();               // Paints controlpanel
    void __fastcall DoScanIRs();             // Scan IRs
    void __fastcall DoScanBumpers();         // Scan Bumpers and paint mobot in environment
    void __fastcall DoErase();               // Delete mobot from environment
```

protected:

```
    void __fastcall SetMobotID(int value);
    int __fastcall GetMobotID();
    void __fastcall SetMobotMoveStep(int value);
    int __fastcall GetMobotMoveStep();
    void __fastcall SetNumberOfIRs(int value);
    int __fastcall GetNumberOfIRs();
    void __fastcall SetIRRange(int value);
    int __fastcall GetIRRange();
    void __fastcall SetIRStep(int value);
    int __fastcall GetIRStep();
    void __fastcall SetNumberOfBumpers(int value);
    int __fastcall GetNumberOfBumpers();
    void __fastcall SetBumperOutCircleRadius(int value);
    int __fastcall GetBumperOutCircleRadius();
    void __fastcall SetBumperQuality(int value);
    int __fastcall GetBumperQuality();
    void __fastcall SetNumberOfTrajectoryPoints(int value);
    int __fastcall GetNumberOfTrajectoryPoints();
```

```
    void __fastcall SetInitialCoordinateX(int value);
    int __fastcall GetInitialCoordinateX();
    void __fastcall SetInitialCoordinateY(int value);
    int __fastcall GetInitialCoordinateY();
    void __fastcall SetInitialDirection(int value);
    int __fastcall GetInitialDirection();


    void __fastcall Paint();              // synchronized paint of controlpanel
    void __fastcall ScanIRs();            // synchronized scan of IRs
    void __fastcall ScanBumpers();  // synchronized scan of Bumpres and paint of mobot in environment
    void __fastcall Erase();              // synchronized erase of mobot from environment

    void __fastcall StepForward();
    bool __fastcall TryStepForward();
    void __fastcall Turn(float value);

    virtual void __fastcall Execute();
    virtual void __fastcall MoveMobot();


    __property int CoordinateX  = { read = GetCoordinateX, write = SetCoordinateX };
    __property int CoordinateY  = { read = GetCoordinateY, write = SetCoordinateY };
    __property float Direction  = { read = GetDirection, write = SetDirection };

    __property int *IRState = { read = FIRState };
    __property int *BumperState = { read = FBumperState };
    __property bool SomeIRIsActive = { read = FSomeIRIsActive };
    __property bool SomeBumperIsActive = { read = FSomeBumperIsActive };
    __property float MobotTurnAngleStep = { read = FMobotTurnAngleStep };
    __property int MobotTurnAngleStepsToPI = { read = FMobotTurnAngleStepsToPI };
```

public:

```
    __fastcall TMobotThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel, TImage
*Environment);
    __fastcall ~TMobotThread();

    void __fastcall LoadDefaultParameters(); // load mobot's parameters from file MobotXX.cfg, where
                                                XX is FMobotID
    void __fastcall SaveDefaultParameters(); // save mobot's parameters to file MobotXX.cfg, where XX is
                                                FMobotID

    void __fastcall SuspendBeforeParametersChange(); // suspend in time acceptable for change
parameters
    void __fastcall ResumeAfterParametersChange();   // resume

    __property int **Trajectory  = { read = FTrajectory };

    __property int MobotID  = { read = GetMobotID, write = SetMobotID };
    __property int MobotMoveStep  = { read = GetMobotMoveStep, write = SetMobotMoveStep };
    __property int NumberOfIRs  = { read = GetNumberOfIRs, write = SetNumberOfIRs };
    __property int IRRange  = { read = GetIRRange, write = SetIRRange };
    __property int IRStep  = { read = GetIRStep, write = SetIRStep };
    __property int NumberOfBumpers  = { read = GetNumberOfBumpers, write = SetNumberOfBumpers
};
    __property int BumperOutCircleRadius  = { read = GetBumperOutCircleRadius, write =
          SetBumperOutCircleRadius };
    __property int BumperQuality  = { read = GetBumperQuality, write = SetBumperQuality };
```

```
    __property int NumberOfTrajectoryPoints  = { read = GetNumberOfTrajectoryPoints, write =
        SetNumberOfTrajectoryPoints };
    __property int InitialCoordinateX  = { read = GetInitialCoordinateX, write = SetInitialCoordinateX };
    __property int InitialCoordinateY  = { read = GetInitialCoordinateY, write = SetInitialCoordinateY };
    __property int InitialDirection  = { read = GetInitialDirection, write = SetInitialDirection };

    __property bool PresenceInEnvironment  = { read = FPresenceInEnvironment, write =
        FPresenceInEnvironment };
    __property bool NotAlone  = { read = FNotAlone, write = FNotAlone };
    __property bool ControlPanelLights  = { read = FControlPanelLights, write = FControlPanelLights };
    __property bool ControlPanelIRMeasuredValues  = { read = FControlPanelIRMeasuredValues, write =
        FControlPanelIRMeasuredValues };
    __property bool ControlPanelOnlyLastActivation  = { read = FControlPanelOnlyLastActivation, write
        = FControlPanelOnlyLastActivation };
    __property bool TrajectoryRecord  = { read = FTrajectoryRecord, write = FTrajectoryRecord };
    __property TColor ColorOfMobot  = { read = FColorOfMobot, write = FColorOfMobot };
    __property TColor ColorOfMobotID  = { read = FColorOfMobotID, write = FColorOfMobotID };
    __property TColor ColorOfFloor  = { read = FColorOfFloor, write = FColorOfFloor };
    __property TColor ColorOfActivatedSensors  = { read = FColorOfActivatedSensors, write =
        FColorOfActivatedSensors };
    __property TColor ColorOfInActivatedSensors  = { read = FColorOfInActivatedSensors, write =
        FColorOfInActivatedSensors };

};
//---------------------------------------------------------------------------
#endif
```

Příloha 3: Definice třídy TAssociativeSearchNetwork

```
//---------------------------------------------------------------------------
#ifndef AssociativeSearchNetworkH
#define AssociativeSearchNetworkH
//---------------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
#include <Math.hpp>
//---------------------------------------------------------------------------
class TAssociativeSearchNetwork
{
private:

   // only fields
   int FASNID;                    // ID of Associative search network
   float* FInputsOld;             // inputs(t-1) array    X(t-1)
   int* FOutputsOld;              // outputs(t-1) array   Y(t-1)
   int* FOutputsOlder;            // outputs(t-2) array   Y(t-2)
   float FPayOffOld;              // reinforcement(t-1)   z(t-1)

   // private properties with set and get methods

   // public properties with direct access ( readonly )
   float* FInputs;                // inputs(t) array      X(t)
   int* FOutputs;                 // outputs(t) array     Y(t)
   float FPayOff;                 // reinforcement(t)     z(t)
   float** FWeights;              // weights(t) matrix    W(t)
   int FCurrentLearningStep;      // number of steps with learning till this time
   // public properties with direct access
   bool FUseOutputsOlder;         // determine if learning rule contains y(t-1)-y(t-2) instead of y(t-1)
   bool FLearning;                // determine if ASN adjust it's weights
   bool FUsingContext;            // determine if ASN use context information X(t)
   // public properties with set and get methods
   int FNumberOfInputs;           // number of ASN inputs (without w0j) = context
   int FNumberOfOutputs;          // number of ASN outputs
   float FNoiseMean;              // noise mean
   float FNoiseDeviation;         // noise deviation
   float FLearningSpeed;          // learning speed                c
   float FLearningSpeedForBiases; // learning speed for biases (w0j) c0
   float FBoundForBiases;         // bound for biases weights ( insure that down-gradient moves can
                                  return the weight to zero )
   bool FUseBiases;               // determine if biases weights are used

   float FMaxChangeInPayoff;      // maximal change in Payoff acceptable to learning ( avoid deadlock
                                  due rapid change in PayOff )
   float FPayOffDivider;          // divider of Payoff allow prove PayOff effect ( PayOff is divided
                                  before learning )

   void __fastcall ArraysAllocateAndInitialize();   //  Allocate and Initialize X(t),X(t-1),Y(t),Y(t-1),
   void __fastcall ArraysDeAllocate();              //      Y(t-2),W(t) according to FNumberOfInputs
                                                    and FNumberOfOutputs
   void __fastcall ClearWeightsMatrix();            //  Clear WeightsMatrix W(t)
   void __fastcall PrepareForNextStep();            //  Stores old inputs, output ane payoff values
                                                    X(t-1)=X(t),Y(t-2)=Y(t-1),Y(t-1)=Y(t) a z(t-1)=z(t)
```

```
   float __fastcall Bound(float value);                  //  Bounds value to the interval [0,
FBoundForBiases]
   void __fastcall CalculateOutputs();                   //  Calculate outputs for current X(t) and W(t)
   void __fastcall AdjustWeights();                      //  Adjust WeightMatrix ( learning rule )
```

protected:

```
   void __fastcall SetNumberOfInputs(int value);
   int __fastcall GetNumberOfInputs();
   void __fastcall SetNumberOfOutputs(int value);
   int __fastcall GetNumberOfOutputs();
   void __fastcall SetNoiseMean(float value);
   float __fastcall GetNoiseMean();
   void __fastcall SetNoiseDeviation(float value);
   float __fastcall GetNoiseDeviation();
   void __fastcall SetLearningSpeed(float value);
   float __fastcall GetLearningSpeed();
   void __fastcall SetLearningSpeedForBiases(float value);
   float __fastcall GetLearningSpeedForBiases();
   void __fastcall SetBoundForBiases(float value);
   float __fastcall GetBoundForBiases();
   void __fastcall SetUseBiases(bool value);
   bool __fastcall GetUseBiases();
   void __fastcall SetMaxChangeInPayoff(float value);
   float __fastcall GetMaxChangeInPayoff();
   void __fastcall SetPayOffDivider(float value);
   float __fastcall GetPayOffDivider();
```

public:

```
   __fastcall TAssociativeSearchNetwork(int ID);
   __fastcall TAssociativeSearchNetwork(int ID, int XCount, int YCount);
   __fastcall TAssociativeSearchNetwork(int ID, int XCount, int YCount, float Mean, float Deviation,
        float C, float Co, float Bound, bool Biases, bool Older, bool Learn, float MaxChangeZ, float
        ZDivider, bool UsingX);
   __fastcall ~TAssociativeSearchNetwork();

   void __fastcall Initialize(int XCount, int YCount, float Mean, float Deviation, float C, float Co, float
        Bound, bool Biases, bool Older, bool Learn, float MaxChangeZ, float ZDivider, bool UsingX);//
        Initialize ASN
   void __fastcall Initialize(int XCount, int YCount); // Initialize ASN

   virtual void __fastcall LoadDefaultParameters();    // load ASN's parameters from file ASNxx.cfg,
                                                       where xx is FASNID
   virtual void __fastcall SaveDefaultParameters();    // save ANS's parameters to file ASNxx.cfg, where
                                                       xx is FASNID

   void __fastcall LoadWeightsMatrix(AnsiString FileName); // load WeightsMatrix W(t) from file
   void __fastcall SaveWeightsMatrix(AnsiString FileName); // save WeightsMatrix W(t) to a file

   void __fastcall ForgetAllYouWasLearn();                  // Clear WeightsMatrix W(t)

   virtual void __fastcall GenerateOutputs(float* Xt, int* Yt);            // Generate output wihout learning
   virtual void __fastcall GenerateOutputs(float* Xt, float Zt, int* Yt); // Generate output and learn (
                                                                          adjust WeightMatrix )


   __property float **WeightsMatrix  = { read = FWeights };
   __property float *Inputs  = { read = FInputs };
   __property int *Outputs  = { read = FOutputs };
   __property float PayOff  = { read = FPayOff };
```

```
    __property int CurrentLearningStep  = { read = FCurrentLearningStep };


    __property int NumberOfInputs  = { read = GetNumberOfInputs, write = SetNumberOfInputs };
    __property int NumberOfOutputs  = { read = GetNumberOfOutputs, write = SetNumberOfOutputs };
    __property float NoiseMean  = { read = GetNoiseMean, write = SetNoiseMean };
    __property float NoiseDeviation  = { read = GetNoiseDeviation, write = SetNoiseDeviation };
    __property float LearningSpeed  = { read = GetLearningSpeed, write = SetLearningSpeed };
    __property float LearningSpeedForBiases  = { read = GetLearningSpeedForBiases, write =
          SetLearningSpeedForBiases };
    __property float BoundForBiases  = { read = GetBoundForBiases, write = SetBoundForBiases };
    __property bool UseBiases  = { read = GetUseBiases, write = SetUseBiases };
    __property float MaxChangeInPayoff  = { read = GetMaxChangeInPayoff, write =
SetMaxChangeInPayoff };
    __property float PayOffDivider  = { read = GetPayOffDivider, write = SetPayOffDivider };

    __property bool UseOutputsOlder  = { read = FUseOutputsOlder, write = FUseOutputsOlder };
    __property bool Learning  = { read = FLearning, write = FLearning };
    __property bool UsingContext  = { read = FUsingContext, write = FUsingContext };

};
//---------------------------------------------------------------------------
#endif
```

Příloha 4: Definice třídy TSimpleMobotThread

```
//---------------------------------------------------------------------------
#ifndef SimpleMobotThreadH
#define SimpleMobotThreadH
//---------------------------------------------------------------------------
#include "ClassMobotThread.h"
//---------------------------------------------------------------------------
class TSimpleMobotThread : public TMobotThread
{
private:

protected:
   virtual void __fastcall MoveMobot();



public:
   __fastcall TSimpleMobotThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel, TImage
        *Environment);
   __fastcall ~TSimpleMobotThread();

};
//---------------------------------------------------------------------------
#endif
```

Příloha 5: Definice třídy TMobotASNThread

```
//---------------------------------------------------------------------------
#ifndef ClassMobotASNThreadH
#define ClassMobotASNThreadH
//---------------------------------------------------------------------------
#include "ClassMobotThread.h"
#include "AssociativeSearchNetwork.h"
//---------------------------------------------------------------------------
class TMobotASNThread : public TMobotThread
{
private:

   // only fields
   TMultiReadExclusiveWriteSynchronizer *FOwnPositionSynchronizer; //  Synchronizer for write Own
                                                                       Position to
   int *FOwnPositionXSaveToInt;            //  communication table (following two pointers to integers)
   int *FOwnPositionYSaveToInt;

   // private properties with set and get methods

   // protected properties with get methods ( readonly )

   // public properties with direct access ( readonly )
   // public properties with direct access
   TMultiReadExclusiveWriteSynchronizer *FTargetPositionSynchronizer; //  Synchronizer for read
                                                                          Target Position to
   int *FTargetXLoadFromInt;               //  communication table (following two pointers to integers)
   int *FTargetYLoadFromInt;
   // public properties with set and get methods


   void __fastcall ContextAllocateAndInitialize();    // Allocate and initialize Context array X(t)
   void __fastcall ContextDeAllocate();               // Deallocate Context array X(t)
   void __fastcall ActionAllocateAndInitialize();     // Allocate and initialize Action array Y(t)
   void __fastcall ActionDeAllocate();                // Deallocate Action array Y(t)

   void __fastcall SetTargetVisibility(int value);
   int __fastcall GetTargetVisibility();

   virtual void __fastcall SetMobotMoveStep(int value);// Extend
                                                          TMobotComponent::SetMobotMoveStep about
                                                          Context array Allocation and Initialization

protected:

   // only fields
   int FTargetCoordinateX;                            // Target coordinate
   int FTargetCoordinateY;

   // public properties with direct access ( readonly )
   float *FContext;                                   // X(t)
   int *FAction;                                      // Y(t)
   float FPayOff;                                     // z(t)
   int FContextArraySize;                             // Size of X(t)
   int FActionArraySize;                              // Size of Y(t)
   TAssociativeSearchNetwork *FASN;                   // ASN object
   // public properties with direct access
```

```
    bool FPayOffInverted;                       // Invert PayOff Flag
    // public properties with set and get methods
    int FTargetVisibility;                      // Distance to which Context and PayOff can be
calculated

    void __fastcall ContextClear();             // Clear Context array X(t)
    void __fastcall PublishOwnPosition();       // Write Own Position to communication table
    void __fastcall CheckTargetPosition();      // Read Target Position from communication table

    virtual void __fastcall ComputeContextAndActionArraySize(); // Compute Context Array Size and
                                                                    Action Array Size
    virtual void __fastcall ComputeContext();   // Evaluate Context X(t)
    virtual void __fastcall ComputePayOff();    / Evaluate PayOff z(t)
    virtual float __fastcall ActionToTurnAngle();   // Return TurnAngle for next step according to
Action Y(t)

    virtual void __fastcall MoveMobot();        // Main Method Control Movement of Mobot
```

public:

```
    __fastcall TMobotASNThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel, TImage
        *Environment, TMultiReadExclusiveWriteSynchronizer *Own, int *OwnX, int *OwnY,
                    TMultiReadExclusiveWriteSynchronizer *Target, int *TargetX, int *TargetY);
    __fastcall ~TMobotASNThread();

    __property TAssociativeSearchNetwork* ASN  = { read = FASN };
    __property float *Context  = { read = FContext };
    __property int *Action  = { read = FAction };
    __property float PayOff  = { read = FPayOff };
    __property int ContextArraySize  = { read = FContextArraySize };
    __property int ActionArraySize  = { read = FActionArraySize };

    __property TMultiReadExclusiveWriteSynchronizer *TargetPositionSynchronizer  = { read =
        FTargetPositionSynchronizer, write = FTargetPositionSynchronizer };
    __property int *TargetXLoadFromInt  = { read = FTargetXLoadFromInt, write =
FTargetXLoadFromInt };
    __property int *TargetYLoadFromInt  = { read = FTargetYLoadFromInt, write =
FTargetYLoadFromInt };
    __property bool PayOffInverted  = { read = FPayOffInverted, write = FPayOffInverted };

    __property int TargetVisibility  = { read = GetTargetVisibility, write = SetTargetVisibility };

};
//---------------------------------------------------------------------------
#endif
```

Příloha 6: Definice třídy TFirstMobotASNThread

```
//---------------------------------------------------------------------------
#ifndef FirstMobotASNThreadH
#define FirstMobotASNThreadH
//---------------------------------------------------------------------------
#include "ClassMobotASNThread.h"
//---------------------------------------------------------------------------
class TFirstMobotASNThread : public TMobotASNThread
{
private:

protected:
   virtual void __fastcall ComputeContextAndActionArraySize(); // Compute Context Array Size and
                                                                Action Array Size
//    virtual void __fastcall ComputeContext();          // Evaluate Context X(t)
//    virtual void __fastcall ComputePayOff();           // Evaluate PayOff z(t)
//    virtual float __fastcall ActionToTurnAngle();      // Return TurnAngle for next step according to
Action Y(t)

//    virtual void __fastcall MoveMobot();               // Main Method Control Movement of Mobot

public:
   __fastcall TFirstMobotASNThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel, TImage
        *Environment, TMultiReadExclusiveWriteSynchronizer *Own, int *OwnX, int *OwnY,
        TMultiReadExclusiveWriteSynchronizer *Target, int *TargetX, int *TargetY);
   __fastcall ~TFirstMobotASNThread();

};
//---------------------------------------------------------------------------
#endif
```

Příloha 7: Definice třídy TSimpleIRusedMobotASNThread

```
//---------------------------------------------------------------------------
#ifndef SimpleIRusedMobotASNThreadH
#define SimpleIRusedMobotASNThreadH
//---------------------------------------------------------------------------
#include "ClassMobotASNThread.h"
//---------------------------------------------------------------------------
class TSimpleIRusedMobotASNThread : public TMobotASNThread
{
private:

protected:
   virtual void __fastcall ComputeContextAndActionArraySize(); // Compute Context Array Size and
                                                                Action Array Size
//    virtual void __fastcall ComputeContext();          // Evaluate Context X(t)
//    virtual void __fastcall ComputePayOff();           // Evaluate PayOff z(t)
//    virtual float __fastcall ActionToTurnAngle();      // Return TurnAngle for next step according to
Action Y(t)

   virtual void __fastcall MoveMobot();                 // Main Method Control Movement of Mobot

public:

   __fastcall TSimpleIRusedMobotASNThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel,
        TImage *Environment, TMultiReadExclusiveWriteSynchronizer *Own, int *OwnX, int *OwnY,
        TMultiReadExclusiveWriteSynchronizer *Target, int *TargetX, int *TargetY);
   __fastcall ~TSimpleIRusedMobotASNThread();

};
//---------------------------------------------------------------------------
#endif
```

Příloha 8: Definice třídy TIRusedMobotASNThread

```
//---------------------------------------------------------------------------
#ifndef IRusedMobotASNThreadH
#define IRusedMobotASNThreadH
//---------------------------------------------------------------------------
#include "ClassMobotASNThread.h"
//---------------------------------------------------------------------------
class TIRusedMobotASNThread : public TMobotASNThread
{
private:

protected:
   virtual void __fastcall ComputeContextAndActionArraySize(); // Compute Context Array Size and
                                                                Action Array Size
   virtual void __fastcall ComputeContext();          // Evaluate Context X(t)
//    virtual void __fastcall ComputePayOff();         // Evaluate PayOff z(t)
//    virtual float __fastcall ActionToTurnAngle();    // Return TurnAngle for next step according to
Action Y(t)

   virtual void __fastcall MoveMobot();               // Main Method Control Movement of Mobot

public:

   __fastcall TIRusedMobotASNThread(bool CreateSuspended,int ID, TPaintBox *ControlPanel, TImage
        *Environment, TMultiReadExclusiveWriteSynchronizer *Own, int *OwnX, int *OwnY,
        TMultiReadExclusiveWriteSynchronizer *Target, int *TargetX, int *TargetY);
   __fastcall ~TIRusedMobotASNThread();

};
//---------------------------------------------------------------------------
#endif
```